Virtools Dev User Guide

Virtools - The Behavior Company

www.virtools.com

Virtools Dev User Guide

Copyright © 2004 Virtools SA

All Rights Reserved

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Virtools.

VIRTOOLS SA (Virtools) PROVIDES THIS MATERIAL "AS IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Virtools may make improvements and changes to the product described in this document at any time without notice. Virtools assumes no responsibility for the use of the product or this document except as expressly set forth in the applicable Virtools license agreement or license agreements and subject to the terms and conditions set forth therein and applicable Virtools policies and procedures. This document may contain technical inaccuracies or typographical errors. Periodic changes are made to the information contained herein: these changes will be incorporated in new editions of the manual.

Virtools is a registered trademark.

Windows and Internet Explorer are registered trademarks of Microsoft, Navigator and Communicator are registered trademarks of Netscape Communications. All other brand or product names are the trademarks or registered trademarks of their respective holders.

Written and edited by Christopher Mc Carthy and David Callele. Reviewed by Florian Delizy and Frédérique Krupa. Cover art by Etienne Mineur

Thanks to Ouali Chabi, Nicolas Philippon, and Idenao (G. Lancrey and R. Enjalbert) for help with tutorials, to Erwann Surcouf and Boris Duong for line art, and to Severine Deneau for original document design.

PART 1 - INTRODUCING VIRTOOLS DEV 11

1 What is Virtools Dev? 13
1.1 An Authoring Application 13
1.2 A Behavioral Engine (CK2) 13
1.3 A Render Engine 14
1.4 A Web Player 14
1.5 A Software Development Kit 14

2 About this User Guide 17
2.1 Introducing Virtools Dev 17
2.2 Looking at Virtools Dev 17
2.3 3D Space in Virtools Dev 18
2.4 Understanding Virtools Dev 18
2.5 Authoring in Virtools Dev 18
2.6 Appendix 19

3 Installing Virtools Dev 21
3.1 Hardware 21
3.2 Software 21
3.3 Installing Virtools Dev 21

4 Getting More Information 23
4.1 Virtools Dev Authoring Application 23
4.1.1 Screentips 23
4.1.2 Online Reference 23
4.2 SDK Documentation 24
4.3 Virtools MiniSite 24
4.4 Internet Based Resources 25

5 Document Conventions 27
5.1 Organization 27
5.2 Emphasis 27
5.3 Lists 28
5.4 Figures 28
5.5 Acronyms 28
5.6 Building Blocks (BBs) 29
5.7 Parameter Operations (paramOps) 29
5.8 Parameter Names (pNames) 30
5.9 Parameter Types (pTypes) 30
5.10 Parameter Values (pValues) 30
5.11 File, Resource and Object Names 30
5.12 Graphical User Interface (GUI) Elements 31
5.13 Attributes 31

PART 2 - LOOKING AT VIRTOOLS DEV . . . 33

6 Virtools Dev at Start-Up 35

7 Menu Bar 37

8 3D Layout 39
8.1 Top Toolbar 39
8.2 Left Toolbar 40
8.2.1 Selection Tools 40
8.2.2 Transformation Tools 40

8.2.3 Reference and Screen Guides 42
8.2.4 Creation Tools. 42
8.2.5 Camera Navigation Tools 43

9 Building Blocks and Data Resources. . . . 45

10 Level Manager. 47

10.0.1 Top Toolbar 47
10.0.2 Left Toolbar 48

11 Schematic 49

11.0.1 Top Toolbar 50

12 Status Bar 53

PART 3 - 3D SPACE IN VIRTOOLS DEV . 55

13 Virtools Dev and 3D Graphics 57

14 Coordinate Systems 59

14.1 The 2D Coordinate System. 59
14.2 The 3D Coordinate System. 60
14.2.1 Units of Measurement. 61
14.2.2 3D Coordinates 62
14.2.3 Vectors 62
14.2.4 Local Coordinate System, Relative Coordinate System. 63
14.2.5 World Coordinate System, Absolute Coordinate System. 64

14.2.6 Referential Axes, Referential 64
14.2.7 Orientation 65
14.2.8 Orientation and the Referential 66

15 Transformations 69

16 Matrix Operations 71

17 Worlds and Levels, Places and Scenes . . 73

18 Cameras and Rendering 75

18.1 Depth of Field, Z buffering 75

19 The Render Engine 79

19.1 The Render Engine – CK2_3D 79
19.2 Virtools Dev Rasterizers 79

PART 4 - UNDERSTANDING VIRTOOLS DEV 81

20 Elements, Classes, and Object Oriented Design 83

20.1 Object Oriented Design 83
20.2 Inheritance 85
20.3 Specialization 86
20.4 Aggregation 86
20.4.1 Run-time Aggregation, The Scene Hierarchy 88

20.4.2 Sharing Elements 88
20.5 Association 89
21 The Elements of a Composition (CMO) . 91
21.1 The Behavioral Object (BeObject) 92
21.2 The Level 92
21.3 Scenes 93
21.4 Places and Portals 94
21.5 Abstract Elements 95
21.5.1 Groups 95
21.5.2 Arrays 95
22 The Virtools Dev Process Loop 97
22.1 Processing Behaviors 99
22.2 Rendering 100
23 The Behavioral Engine 101
23.1 Behavior Loops 103
23.2 Priority 104
24 Behaviors and Scripts 107
24.1 Behavior Building Block (BB) 107
24.1.1 Interpreting a BB Symbol 107
24.1.2 Behavior Input, bIn 108
24.1.3 Behavior Output, bOut 108
24.1.4 Behavior Link, bLink 108
24.1.5 Parameter Input and Parameter Output 109
24.1.6 Target Parameter 109
24.1.7 C, S, and V 111
24.1.8 Messages 113

24.1.9 BB Processing 113
24.2 Behavior Graph (BG) 114
25 Parameters 117
25.1 Parameter Types 117
25.2 Parameter Input, pIn 118
25.3 Parameter Output, pOut 118
25.4 Parameter Link, pLink 119
25.5 Local Parameter 119
25.6 This 119
25.7 Parameter Shortcuts 120
26 Parameter Operations (paramOps) . . . 121
26.1 Parameter Notation 122
26.2 paramOps and Behaviors 123
26.3 Advanced paramOps 124
26.3.1 Order of pIns 124
26.3.2 Calculating a Value at a Specific Moment 124
26.3.3 pTypes <Angle>, <Float>, and <Percentage> 125
27 Attributes 127
27.1 Attribute Shortcuts 128

PART 5 - AUTHORING IN VIRTOOLS DEV ... 129

28 Quick Start 131

28.1 Overview 131
28.2 Organize Resources 131
28.3 Plan the Content 132
28.4 Import Media 132
28.4.1 Importing the Scenery 132
28.4.2 Exploring the Scene in Author Mode 134
28.4.3 Adding a Character and Animations 134
28.5 Arrange the Scene 135
28.5.1 Adding a Camera 136
28.5.2 Activating the Camera at the Start of the Scene 137
28.5.3 Targeting the Camera 138
28.6 Add Interactivity 139
28.6.1 Controlling the Character 139
28.6.2 Adding Keyboard Support 140
28.7 Test 141
28.7.1 Switching to Play Mode 141
28.7.2 Returning to Author Mode 141
28.8 Refine 142
28.8.1 Making the Character Stay on the Floor 142
28.8.2 Adding Simple Collision Management 144
28.8.3 Declaring Objects as Obstacles 144
28.9 Test Again 145
28.9.1 Switching to Play Mode 145

28.9.2 Returning to Author Mode 145
28.10 Refine Again 146
28.10.1 Dynamically Switching Cameras 146
28.11 Test Again 148
28.11.1 Switching to Play Mode 148
28.11.2 Returning to Author Mode 149
28.12 One Last Refinement 149
28.13 Export Content 149
28.13.1 Saving Your Hard Work 149
28.13.2 Sharing Your Content With Others 150
28.14 If You Encountered Any Difficulties... 150
28.15 Congratulations 151

29 Particles 153

29.1 Introduction 153
29.2 Emitters 153
29.3 Particles 154
29.4 Deflectors 154
29.5 Interactors 154
29.6 Configuring a Particle System 155
29.7 StartCmos and FinishedCmos 157
29.8 Exercise 1 - Particle System Basics 158
29.8.1 Start 158
29.8.2 Placing an Emitter 158
29.8.3 First Few Particles 159
29.8.4 Changing a Basic Parameter: Adding Color 160
29.8.5 Texturing Particles 160
29.8.6 Configuring Speed, Lifespan, and Size 161
29.8.7 Variance and Other Parameters 162

29.8.8 Conclusion 162
29.9 Exercise 2 - Moving an Emitter and Adding Interactors 162
29.9.1 Creating a Moving Emitter 163
29.9.2 Adjusting Gravity 164
29.9.3 Adding Wind. 165
29.9.4 Multiple Attributes on a Single 3D Frame 166
29.9.5 Magnet and Other Attributes 167
29.9.6 Conclusion 168
29.10 Exercise 3 - Deflectors 168
29.10.1 Start 169
29.10.2 Creating a Deflector 169
29.10.3 Placing and Resizing a Deflector 170
29.10.4 Placing a Deflector on an Object 171
29.10.5 Editing Particle System Attributes 171
29.10.6 Conclusion 172
29.11 Exercise 4 - Using Animated Textures with Particles 172
29.12 Exercise 5 - Creating 3D Particles 174
29.12.1 Conclusion 175
29.13 Multiple Particle Systems 175
29.14 Frame Rates with Particle Systems. 176

PART 6 - APPENDIX 179

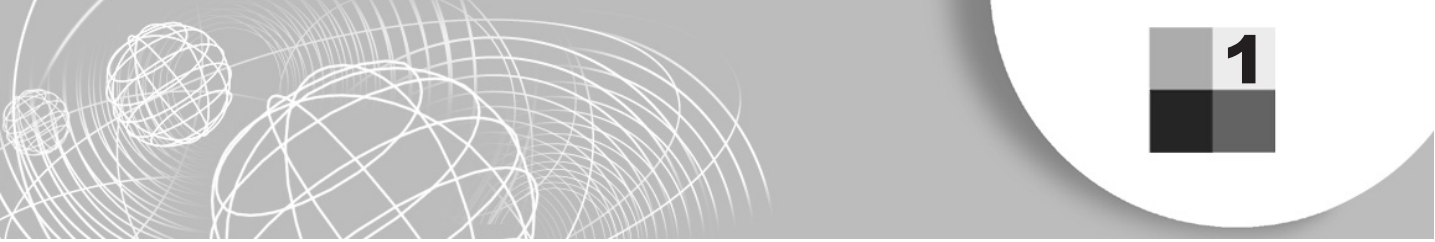
30 Glossary 181
30.1 How to Use the Glossary 181

30.2 Definitions and CKClasses 182
30.3 Terms and Definitions 184

31 Controlling the Orientation of an Element . 217

32 Example Transformations. 219

32.1 Translation 220
32.2 Rotation 220
32.2.1 About Local X Axis 220
32.2.2 About Local Y Axis 221
32.2.3 About Local Z Axis 221
32.3 Scale 222



Part 1 - Introducing Virtools Dev

Welcome to Virtools Dev. Start here and we should have you up and running in no time at all.

Part 1 contains:

1 What is Virtools Dev? - a short description of the components that make up Virtools Dev

2 About this User Guide - an overview of this user guide

3 Installing Virtools Dev - installation requirements and installation procedures

4 Getting More Information - how and where to find further information

5 Document Conventions - how to interpret the style guidelines used in this User Guide

1 WHAT IS VIRTOOLS DEV?

Virtools Dev is an extensive collection of technologies for 3D visualization. The Virtools Dev technologies are broadly grouped as:

1. an Authoring application
2. a Behavioral Engine (CK2)
3. a Rendering Engine
4. a Web Player
5. a Software Development Kit (SDK)

1.1 An Authoring Application

Virtools Dev is an authoring application that allows you to quickly and easily create *compositions* (CMOs) full of rich, interactive, 3D content. Industry standard media such as models, animations, images and sounds are brought to life by Virtools' behavior technologies.

You cannot create models in Virtools Dev; Virtools Dev is not a modeling application. However, simple elements such as Cameras, Lights, Curves, interface elements, and 3D Frames (called dummies or helpers in most 3D modeling applications) can be created with the click of an icon.

1.2 A Behavioral Engine (CK2)

Virtools Dev is a behavioral engine – that is, Virtools Dev processes *behaviors*. A behavior is simply a description of how a certain element acts in an environment. Virtools Dev provides an extensive collection of reusable behaviors in the form of Behavior Building Blocks (BBs) that allow you to create almost any type of content through a simple, graphical interface – without writing a single line of code!

Virtools Dev also has a number of *managers* that help the Behavioral Engine

perform its duties. Some of these managers (such as the TimeManager) are an internal part of the Behavioral Engine while others (such as the SoundManager) are external to the Behavioral Engine.

Virtools Dev's Behavioral Engine is often referred to as CK2, the name of the central software component.

1.3 A Render Engine

Virtools Dev is a render engine that draws the image you see in 3D Layout. The Virtools render engine can be replaced with a render engine of your own, or customized to fit your specific needs using the Software Development Kit (SDK).

1.4 A Web Player

Good technology must be easily accessible before it can be considered a great technology.

Virtools provides a free Web Player that can be downloaded by anyone which benefits of small small file size and a worldwide online distribution through AKAMAI, for fast and reliable downloading and rapid installation.

The Web Player contains a Playback only version of the Behavioral Engine and the complete Render Engine.

Further information on the Virtools Web Player is available in the Online Reference (OR).

1.5 A Software Development Kit

Virtools Dev is a Software Development Kit (SDK) that provides access to the Behavioral Engine and the Render Engine. With the SDK, you can

- create new behaviors or modify existing ones

- create new parameter types
- create media plugins to read any media of your choice
- replace the Virtools Dev render engine with the one of your choice
- create a custom executable file (.exe)
- modify and extend the Virtools Dev render engine

These are just a few examples – your creativity is the only limit to how far you can go! (full source code to the rendering engines is provided for reference).

2 ABOUT THIS USER GUIDE

The Virtools Dev User Guide contains five major parts and an Appendix:

1. Introducing Virtools Dev
2. Looking at Virtools Dev
3. 3D Space in Virtools Dev
4. Understanding Virtools Dev
5. Authoring in Virtools Dev
6. Appendix

2.1 Introducing Virtools Dev

Introducing Virtools Dev is the part you are currently reading. This part contains a concise description of the components that make up Virtools Dev and also contains practical information on how to get the most out of this User Guide.

Read this part for the software and hardware requirements for installing Virtools Dev, as well as the installation instructions. This part also contains information on document conventions used in this book.

2.2 Looking at Virtools Dev

Looking At Virtools Dev is a brief guided tour of the Virtools Dev interface. The icons used by Virtools Dev are presented, identified, and briefly described.

If you like to know what the user interface controls mean before you use a product, then this is the part for you.

2.3 3D Space in Virtools Dev

3D Space in Virtools Dev describes how Virtools Dev creates its 3D space and the rules that govern it. Just how can you have a 3D space on a 2D screen?

This part is essential reading for those new to 3D and recommended for those who have been working in 3D for some time. Concentrating on the basic concepts of 3D, and how they are expressed in Virtools Dev, this part starts off nice and easy and before you know it, you are done!

If you can't get enough of vectors and matrices, there are one or two sections in the Appendix that will interest you too.

2.4 Understanding Virtools Dev

Understanding Virtools Dev explains (almost) all you ever wanted to know about Virtools Dev but didn't know who to ask. This part answers questions like: *Is it possible to have more than one level in a composition?* and *Why can some behaviors only be applied to certain elements?*

Whatever your experience level, this part is essential reading for everyone. You could get along working in Virtools Dev without reading this part, but why make things hard for yourself?

2.5 Authoring in Virtools Dev

This is the part that everybody asks about - where can I find tutorials for Virtools Dev? This part contains two tutorials: a Quick Start for those of you just starting out in Virtools Dev, and a fun and informative tutorial on particles for those of you who have some experience with Virtools Dev already.

For the Quick Start, we would prefer you to peruse (and even read) the other parts beforehand. But if you are too impatient, we have taken care to make the Quick Start accessible to everyone and we sincerely hope we don't lose anyone on the way. Just make sure you read the other parts after!

The particles tutorial is just a sample of what is to come. Further tutorials will be posted to the Virtools website on a regular basis and you can even find some previews in the Virtools MiniSite.

The Virtools MiniSite is installed in the Documentation folder. However, if you chose not install this component, you will have to launch the installation program once more, and choose to install the MiniSite only.

2.6 Appendix

At the end of the User Guide, last - but certainly not least - is the Appendix. Our appendix is larger than most - mostly because of our extensive glossary. The Virtools Glossary contains definitions for many Virtools words and terms that newcomers and pros alike will come to appreciate for their clarity and valuable contextual information.

The Glossary is not all there is to the Appendix - you will also find further help for those tricky orientation and transformation questions.

3 INSTALLING VIRTOOLS DEV

To successfully install and use Virtools Dev, please check that you meet the minimum hardware and software requirements listed below.

NOTE Should you need them, Microsoft DirectX and Internet Explorer are also provided on the installation CD.

3.1 Hardware

- Pentium II or equivalent
- 64 MB of RAM
- CD-ROM drive
- Monitor capable of displaying 1024 by 768 in 16 bit color (65536 color/ Hi-color)
- Direct3D or OpenGL compatible 3D graphic accelerator card with 8 MB of RAM
- Pointing device (mouse, trackball, etc.)
- Sound Card (not a requirement but recommended)

3.2 Software

- Microsoft Windows (9x or higher, NT 4.0 SP6 or higher)
- Microsoft Internet Explorer 4.0 or higher (for the Online Reference)
- Microsoft DirectX 5.0 or higher for DirectX compatible 3D graphic accelerator cards

3.3 Installing Virtools Dev

You can choose the components of Virtools Dev that you wish to install. If you choose to install all components, you will need approximately 600 MB of disk space.

1. Insert the Virtools Dev CD
2. Follow the on-screen instructions
3. If your CD-ROM drive does not automatically run the installation program, manually execute Setup.exe in the Dev folder of the Virtools Dev CD-ROM

NOTE If you have previous versions of Virtools software installed on your computer, ensure that you install this version in a new program folder to avoid any potential upgrade problems.

NOTE Virtools Dev uses a new file format that is not backwards-compatible with file formats used by previous versions of Virtools products. However, Virtools Dev can open and use files saved by previous versions of Virtools products.

4 GETTING MORE INFORMATION

There are a number of sources for more information on Virtools and Virtools Dev:

1. the Online Reference help system from within the Virtools Dev authoring application
2. the Virtools Dev SDK and the SDK Help files
3. the Virtools MiniSite
4. Internet based resources

4.1 Virtools Dev Authoring Application

There are two sources of help integrated with the Virtools Dev application: screentips and the Online Reference.

4.1.1 Screentips

Screentips are available for all icons and most buttons in the Virtools Dev interface. A screentip is a text popup that appears when the mouse hovers over an icon or button. Screentips are used by Virtools Dev to display the name of an icon or button, and to display the keyboard shortcut for that icon or button if one exists.

4.1.2 Online Reference

The Virtools Dev Online Reference is in the form of a compiled help file, providing full text search, a comprehensive index, and a favorites tab for bookmarking pages you find useful pages. The Online Reference is divided into these topics:

1. **Interface** - the Virtools Dev interface
2. **Concepts** - the concepts underlying Virtools Dev

3. **Scripting** - documentation for every Building Block (BB) and a list of all parameter operations (paramOps)
4. **Delivering Content** - how to deliver your wonderful compositions to your audience
5. **Importing media** - supported formats, modeling tips and tutorials

4.2 SDK Documentation

The Virtools Dev SDK allows you to extend Virtools Dev in ways that only you can imagine. The Virtools Dev SDK documentation contains a wealth of information on the internal workings of Virtools Dev. The Virtools Dev SDK contains:

1. **SDKHelp.chm** - the most up to date source for detailed information on the Virtools Dev architecture and principles. The SDK help includes many entries that place the components of Virtools Dev in context, numerous commented code samples, and reference entries for the software elements of the Virtools Dev SDK.
2. **VxMathHelp.chm** - the Virtools Dev SDK makes extensive use of mathematical operations. A library of commonly used functions is provided as part of the SDK.
3. **Code samples** - a large library of source code examples is provided, including the complete source code for all BBs and the Virtools Render Engines! Source code samples for file format translators and a standalone Player are also provided.

4.3 Virtools MiniSite

The Virtools MiniSite is a place of learning - and a few well intentioned distractions. It is divided into five parts:

1. **Technical Samples** - BBs in action, some with commented scripts if you decide to open the CMO in Virtools Dev

2. **Sample Resources** - information and suggested uses for some elements contained in the VirtoolsResources data resource
3. **Tutorials** - includes CMOs and AVI video files
4. **Web Player Commands** - how to get the most of the Virtools Web Player
5. **Demo** - presents an imaginary game teaser shown at the Siggraph 2000 trade show - complete with editable CMO so that you can see exactly how it was done!

4.4 Internet Based Resources

There are various websites and mailing lists that can help you learn Virtools Dev and help you to keep informed of the latest Virtools developments:

1. **Virtools website** (www.virtools.com) - for official company news
2. **Virtools newsletter** (join the mailing list from the Virtools website)
3. **The SwapMeet** (www.theswapmeet.com) - a significant resource for the Virtools user community, it includes the ever popular Discussion Forums
4. **User Group** mailing list (join this mailing list from the Swap-Meet website)

5 DOCUMENT CONVENTIONS

This section defines the document conventions used throughout the User Guide.

NOTE You are strongly encouraged to refer to the Glossary whenever you encounter a term that is unfamiliar. Virtools Dev uses a wide variety of terms, some that are unique to this development environment. These terms, and many terms from the field of 3D graphics, are explained in the Glossary.

5.1 Organization

The User Guide is organized into five numbered Parts. Each Part presents an overview of the Sections contained in that Part. Sections, and the topics within each Section, are organized using outline numbering.

For example, a series of Sections with the following numbering indicates that the material is organized and related as stated.

2 Major Topic

2.1 Sub-topic of 2

2.1.1 Sub-topic of 2.1

2.1.2 Sub-topic of 2.1

2.2 Sub-topic of 2

5.2 Emphasis

Emphasis is used to draw your attention to

- a particular point that is important to comprehension of the matter under discussion.

  For example:

  Objects instantiated from these classes are the objects that *can* have behaviors attached to them but they are *not required* to have behaviors.

- the first time a term specific to Virtools Dev is used. You are advised to consult the Glossary for further information on that term.

  For example:

  In Virtools Dev the point of reference is identified as the *referential.*

NOTE Emphasis is also conveyed using notes like this.

5.3 Lists

Numbered lists indicate that the list elements are related in an ordered manner. For example, the list elements may enumerate all possible options or detail a sequence of events.

Bulleted lists indicate that list elements are related but that no special order is implied by the sequence in which the list elements are presented. For example, the list elements may provide a partial list of possible options or suggested applications.

5.4 Figures

A figure is presented in the section that first references the figure. Within a paragraph, a figure reference is presented as shown.

If you look at the Material Setup in **8-4**, you can see...

5.5 Acronyms

Many terms have acronyms in Virtools. The first time a term commonly referred to by an acronym is used in a section, the term is spelled out followed by the acronym in brackets.

- Software Development Kit (SDK)
- Building Block (BB)

Once an acronym is introduced, the acronym is used interchangeably with the

full text in the remainder of the document.

5.6 Building Blocks (BBs)

The first time a BB name is mentioned in a Section, the name of the BB is always followed by the location of the BB in parenthesis. If the user is being directed to attach the BB, then the location of the BB always follows the name of the BB.

> Drag **Rotate** (3D Transformations/Basic) onto the frame in **3D Layout**.

5.7 Parameter Operations (paramOps)

Virtools Dev uses the following notation to describe paramOps in written form.

For binary operations (operations with two pIns), the syntax is

> Result or Output **Operation Type** Input1 Input2

For unary operations (operations with one pIn), the syntax is

> Result or Output **Operation Type** Input1

In all cases, the syntax is strictly defined as

> <pType> **paramOp** vectors

For example:

> <Vector> **Multiply** <Float> <Vector>

represents a **Multiply** Operation that produces an output result of pType <Vector>, using inputs of pType <Float> and pType <Vector>.

Values can also be associated with the pIns and pOut.

> <Float=50.0> **Multiply** <Float=10.0> <Int=5>

5.8 Parameter Names (pNames)

Parameter Names are formatted as shown.

Change the value of **Speed** to...

5.9 Parameter Types (pTypes)

Parameter types are formatted as shown. Parameter types are always bounded by angle brackets.

You should remember that parameter types <Float>, <Angle> and <Percentage> are actually all the same type.

5.10 Parameter Values (pValues)

Parameter values are formatted as shown.

Change the value of **Speed** from 0.001 to 0.0001

5.11 File, Resource and Object Names

File, resource and object names are formatted as shown.

To import myCharacter.3ds as a character, you...

To open the Mesh Setup for Cube, you...

Where relevant, the file location is shown in brackets

Open tutorial.cmo (Documentation/Cmos/StartCmos)

From Virtools Resources, drag myCharacter.3ds (Characters) into the 3D Layout.

5.12 Graphical User Interface (GUI) Elements

All GUI elements such as Editors (Level Manager, Schematic, Parameter Debugger, Dialog Boxes, BB names, paramOps, etc.) are formatted as shown.

From **Virtools Resources**, drag xyz into **3D Layout**.

In the **Edit Parameters** dialog box, select...

Drag **Rotate** (3D Transformations/Basic) onto the 3D frame in **3D Layout**.

5.13 Attributes

All Attributes are formatted as shown.

The frame now has the attribute *Particle Plane Deflector*.

Part 2 - Looking at Virtools Dev

Looking At Virtools Dev is a brief guided tour of the Virtools Dev interface. The icons used by Virtools Dev are presented, identified, and briefly described.

If you like to know what the user interface controls mean before you use a product, then this is the part for you.

6 Virtools Dev at Start-Up - a guided tour of the Virtools Dev interface in its default state

7 Menu Bar - a closer look at the Menu Bar and some of its enhanced features

8 3D Layout - the tools for manipulating your compositions

9 Building Blocks and Data Resources - accessing those amazing reusable resources

10 Level Manager - tools for organizing your compositions

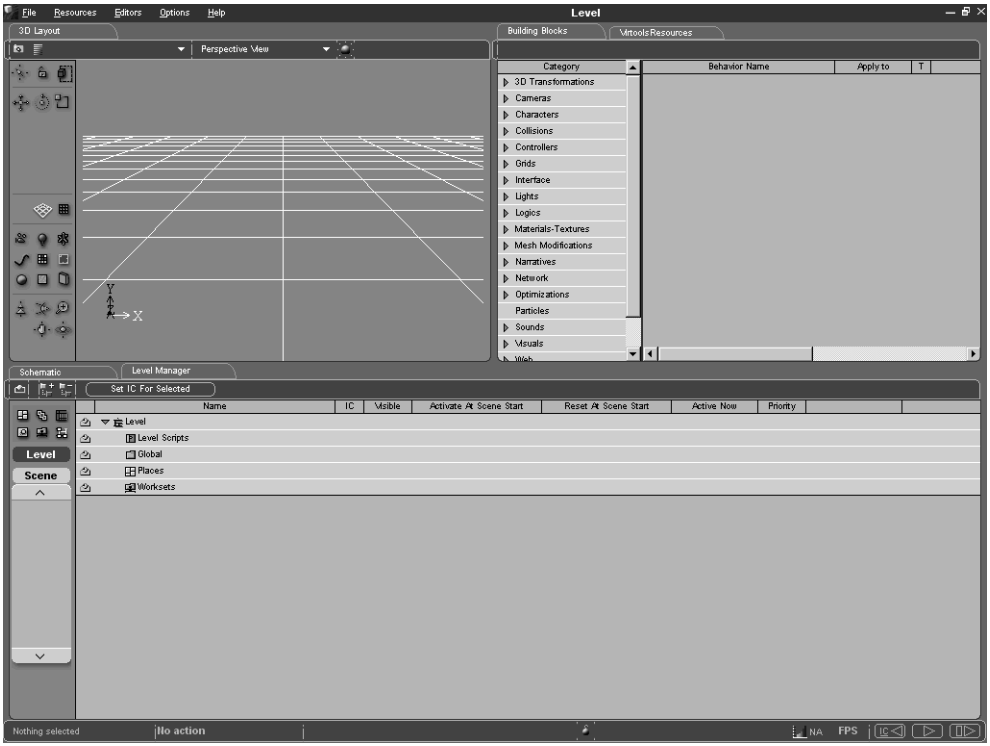
11 Schematic - where you bring your resources to life

12 Status Bar - important messages and controls for testing and refining your compositions

6 VIRTOOLS DEV AT START-UP

Figure **6-1** shows you Virtools Dev at start-up. Virtools Dev uses a tabbed windows system that allows you to move, resize or float windows. The active window is shown by a black tab, all inactive windows have gray tabs. Refer to the Online Reference (Interface) for more information.

6-1 Virtools Dev start-up screen



Virtools Dev is effectively divided into three main regions, with a menu bar at the top and a status bar at the bottom.

The top left region contains the **3D Layout** window, used to visualize your project in real-time. The top right region contains resources in the form of

Building Blocks and **Data Resources**.

The bottom half of the screen contains the **Level Manager** and **Schematic** windows, used for organizing your project and creating scripts respectively.

NOTE All windows in Virtools Dev can be moved from region to region or made to float as separate windows. Due to the dynamic nature of the interface, the User Guide describes the interface with reference to its start up state only.

To return window sizes to their start up state, choose **Refresh Windows** from the **Options** menu.

7 MENU BAR

The menu bar is at the very top of the Virtools Dev screen. There are five menus in Virtools Dev: **File**, **Resources**, **Editors**, **Options** and **Help**.

To the right of the **Help** menu, the menu bar also contains the name of the current scene and the file name and path of the current composition (CMO).

7-1 The Menu bar

Below you will find the name of all Virtools Dev menus followed by short description.

File Menu: file management, open, save and export files.

Resources: resource management, create new data resources, import media.

Editors: additional managers and debuggers.

Options: General Preferences and several useful tools for managing the interface.

Help: the Online Reference and the About... box.

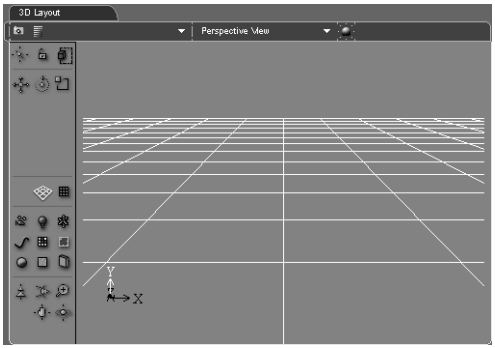
NOTE If NXN AlienBrain is installed on your system, a **NXN alienbrain** Menu will be inserted in between the **Options** and **Help** menus.

Please refer to the Online Reference (Interface) for further information about the options within in each menu.

8 3D LAYOUT

The default position for the **3D Layout** window is the top left region. **3D Layout** consists of a top toolbar, a left toolbar and the rendering window.

8-1 3D Layout



The type of elements and how they appear in the render window depends greatly on the preferences you have set. See the Online Reference for further information (Interface/Menu Bar/Options Menu/General Preferences).

Not all tools are visible at all times. For example, the tools for choosing an axis constraint or planar constraint appear only when you choose either **Select and Translate** or **Select and Rotate**.

Below you can find all the icons contained in the **3D Layout**, each icon with its name and a short description. For further information, including how to use each tool, refer to the Online Reference (Interface/3D Layout).

8.1 Top Toolbar

Snapshot
Captures all of or a portion of the render window.

3D Layout Explorer
Tool for inspecting all elements in 3D Layout.

Selection Group
Choose an existing selection group or create a new selection group.

Select Camera
Displays the current active camera, lists all available cameras.

General Preferences
Opens the General Preferences dialog box.

8.2 Left Toolbar

8.2.1 Selection Tools

Select
Selects elements.

Lock Selection
Toggles lock selection.

Selection Mode
Toggles between normal and strict Selection Mode.

8.2.2 Transformation Tools

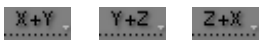
Select and Translate
Moves elements.

Select and Rotate
Rotates elements.

Select and Scale
Scales elements uniformly, volumetrically or normally.

Constrain X, Y or Z Axis
Constrains movement to either the X, Y or Z Axis.

NOTE The Constrain Axis icons only appear when either **Select and Translate** or **Select and Rotate** is active.



Constrain Plane
Constrains movement to the XY, YZ or ZX plane.

NOTE The Constrain Plane icons only appear when either **Select and Translate** or **Select and Rotate** are active.

Referential Axis
Selects the referential for translation and rotation operations in the 3D Layout.

NOTE The Referential Axis icon only appears when either **Select and Translate** or **Select and Rotate** are active.

Toggle Snap
Selects the Snap mode for the movement.

NOTE The Toggle Snap icon only appears when one of the transformation tools is active.

Toggle Hierarchy
Determines if the movement should apply to the selection's children.

NOTE The Toggle Hierarchy icon only appears when one of the transformation tools is active.

Pivot Axis
Determines the pivot point (Object, Local, Selection Center) for a rotation or scale operation.

NOTE The Pivot Axis icon only appears when either **Select and Rotate** or **Select and Scale** are active.

8.2.3 Reference and Screen Guides

Toggle Reference Guides
Shows/hides the 3D Reference Guides.

Toggle Screen Guides
Shows/hides the Screen Guides.

8.2.4 Creation Tools

Create Camera
Creates a new Camera and opens the Camera Setup.

Create Light
Creates a new Light and opens the Light Setup.

Create 3D Frame
Creates a new 3D Frame and opens the 3D Frame Setup.

Create Curve
Creates a new Curve and opens the Curve Setup.

Create Grid
Creates a new Grid and opens the Grid Setup.

Create 2D Frame
Creates a new 2D Frame and opens the 2D Frame Setup.

Create Material
Creates a new Material and opens the Material Setup.

Create Texture
Creates a new Texture and opens the Texture Setup.

Create Portal
Creates a new Portal and opens the Portal Setup.

NOTE If you do not want a Setup to open when you create an entity, change the settings in the **3D Layout - Interface** section of the **General Preferences**. See the Online Reference for further information (Interface/Menu Bar/Options Menu/General Preferences)

8.2.5 Camera Navigation Tools

Camera Dolly
Moves the camera and its target (if any) along the camera's Z axis.

Camera Field of View
Changes the Camera Field of View.

Camera Zoom
Moves the camera toward its target or away from its target (Zoom, Zoom on Selection, Zoom on Scene).

Roll Camera
Rotates the camera about the camera's Z axis.

NOTE The Roll Camera icon only appears when the currently active camera does not have a target.

Camera Pan
Moves the camera and its target (if any) along the camera's XY plane.

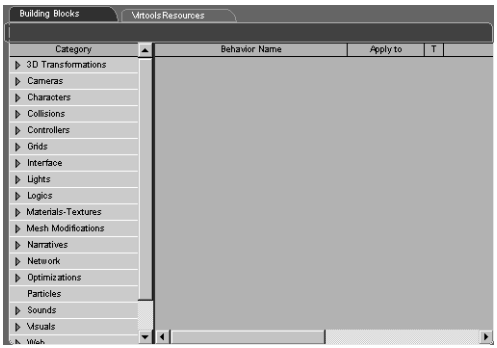
Orbit Target/Orbit Around
Orbit (rotate around) the selection or rotate the camera about its own origin.

9 BUILDING BLOCKS AND DATA RESOURCES

The default position for **Building Blocks** and **Data Resources** is the top right region.

Both resources are organized in a tree structure - click the triangle next to an index in the left pane of the window to expand a branch. The right pane displays the contents of the branch.

9-1 Building Blocks (foreground) and Virtools Data Resources (background)



Building Blocks contains the behaviors provided with Virtools Dev and used by you to make your content interactive. **VirtoolsResources** is the sample **Data Resource** provided with Virtools Dev. **VirtoolsResources** contain media data (with or without scripts attached) and Behavior Graphs in file formats that Virtools Dev can use.

Perhaps the most appropriate analogy is to consider **Building Blocks** as a library of behaviors, organized by the type of behavior and/or by the type of element to which the behavior can be attached. **Data Resources** can be considered project management tools - you organize your media on a per project basis, creating a new **Data Resource** for each project. Of course, you do not have to use **Data Resources** in this way - you are free to keep all of your

media in a single **Data Resource**.

Media data (such as a model, a sound, or an image, etc.) is added to the composition by dragging the media from a **Data Resource** into the **3D Layout** or **Level Manager**.

Building Blocks (BBs), however, cannot be added in the same manner - they must also be attached to a Behavioral Object (BeObject). You can either drag a BB onto a BeObject in the **3D Layout** or **Level View**, or drag a BB directly into a script in the Schematic.

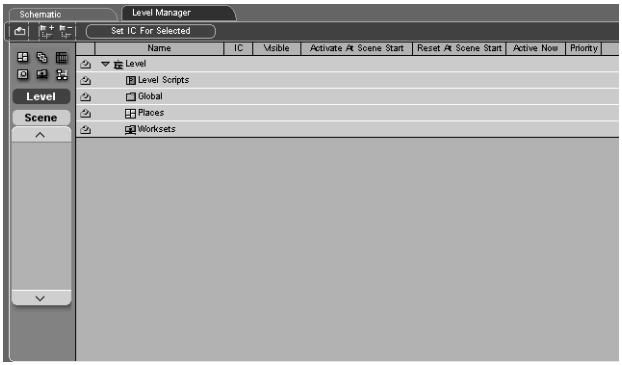
For more information, see the Understanding Virtools Dev part of this User Guide and also refer to the Online Reference (Interface/Building Blocks Resource and Interface/Data Resources).

NOTE If you performed a typical install, the default **Data Resources** are **VirtoolsResources**. You can remove the **VirtoolsResources** from the display (by right clicking on the tab and closing the window). You can also create and add your own **Data Resources**.

10 LEVEL MANAGER

The default position for the **Level Manager** is in the bottom half of the screen. The **Level Manager** consists of a top toolbar, a left toolbar and the main window listing all elements of the Level or Scene by CKClass.

10-1 Level Manager



The **Level Manager** helps you to organize your composition, and contains a wealth of information.

NOTE There are two modes for the **Level Manager**: Level (the default mode) and Scene.

Below you can find all icons and buttons contained in the **Level Manager**, each icon and button with its name and a short description. For further information, including how to use these tools, refer to the Online Reference (Interface/Level Manager).

10.0.1 Top Toolbar

Show/Hide Layer
Shows/hides all items with the Layer flag set.

Expand Selected
Expands the selected branch.

Collapse Selected
Collapses the selected branch.

IC for Selected
Sets Initial Conditions (IC) for the selected elements.

10.0.2 Left Toolbar

Create Place
Creates a new Place.

Create Group
Creates a new Group.

Create Array
Creates a new Array and opens the Array Setup.

Create Scene
Creates a new Scene.

Create Workset
Creates a new Workset or Sub-Workset.

Create Script
Creates a new Script.

Level
Sets the behavioral context to Level mode.

Scene
Sets the behavioral context to Scene mode.

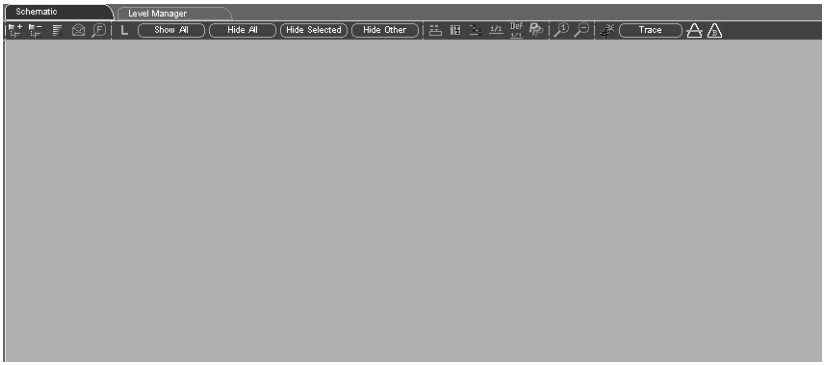
Up
Selects the previous scene.

Down
Selects the next scene.

11 SCHEMATIC

The default position of the **Schematic** is in the bottom half of the screen, behind the **Level Manager**. The **Schematic** consists of a top toolbar and, once you have started creating interactivity, scripts.

11-1 Schematic at start-up



NOTE The **Schematic** usually contains scripts (without scripts, you cannot have any interactivity!). See "Behaviors and Scripts" on page 107 and the Online Reference (Interface/Schematic) for further information on scripts.

The **Schematic** allows you to manage and edit your scripts. Scripts contain the interaction of a composition; scripts visually describe behaviors.

NOTE You do not actually create scripts in the **Schematic**. New scripts are created in the **3D Layout** or **Level Manager** and interactivity is assembled in the **Schematic**.

Below you will find all icons and buttons contained in the **Schematic**, each icon and button followed by its name and a short description. For further information, including how to use these tools, refer to the Online Reference (Interface/Schematic).

11.0.1 Top Toolbar

Expand Selected Scripts
Expends the selected Scripts.

Collapse Selected Scripts
Collapses the selected Scripts.

Schematic Explorer
Opens the **Schematic Explorer** displaying the contents of the **Schematic**. As your compositions grow, the **Schematic Explorer** becomes an increasingly useful tool for managing the elements of your composition

Message Explorer
Opens the **Message Explorer**, displaying a list of messages, senders and receivers.

Find Text
Opens the **Find Text Dialog**, allowing a fast search in the objects names.

Level/Scene Filter
When the **L** button is actve there is no filter for shown scripts int schematic. When the **S** button is active, only scripts of active scene are shown.

Show All
Displays all Scripts.

Hide All
Hides all Scripts.

Hide Selected
Hides selected Scripts.

Hide Other
Hides unselected Scripts.

Show/Hide Local Parameters
Toggles display of local parameters.

Show/Hide Script Header
Toggles display of the script header (the left part of the script).

Show/Hide Control Points
Toggles display of link control points.

Show/Hide Link Delays
Toggles display of link delays.

Edit Default Link Delay
Edits the default link delay.

Show/Hide Priorities
Toggles display of priorities.

Reset Schematic Zoom and Position
Resets the Schematic zoom factor to 1.

Zoom Mode
Modifies the zoom factor.

Script Debugger
Opens the **Script Debugger**.

Trace Mode
Toggles trace mode.

Link with Errors explorer
Opens the **Link with Errors after Loading Explorer**.

Obsolete BBs explorer
Opens the **Obsolete BBs Explorer**.

12 STATUS BAR

The **Status Bar** is at the very bottom of the Virtools Dev screen.

12-1 Status Bar

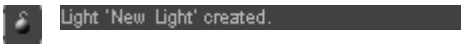
The **Status bar** displays useful, context-dependent information. The **Status Bar** also contains five icons.

Below you will find an image of these information fields or icons, followed by the name and a short description. For further information, including how to use these tools, refer to Online Reference (Interface/Status Bar).

Selection
Displays the name of selected element.

Action
Displays the name of the action you are performing.

Coordinates
Displays the XYZ coordinates of a selected object.



Event Log
Opens the Event Log. Ongoing Event Log messages are displayed next to the Event Log icon as they occur.

Profiler
Opens the Profiler.

FPS (Frames Per Second)
Displays the current number of frames per second; displays NA while in Author mode and updates continually in Play mode.

Reset IC (Initial Conditions)
Resets all Initial Conditions.

Play/Pause
Plays or pauses the composition.

Advance One Step
Causes the behavioral engine to process one frame only.

Part 3 - 3D Space in Virtools Dev

3D Space in Virtools Dev describes how Virtools Dev creates its 3D space and the rules that govern it. Just how can you have a 3D space on a 2D screen?

This part is essential reading for those new to 3D and recommended for those who have been working in 3D for some time. Concentrating on the basic concepts of 3D, and how they are expressed in Virtools Dev, this part starts off nice and easy and before you know it, you are done!

If you can't get enough of vectors and matrices, there are one or two sections in the Appendix that will interest you too.

13 Virtools Dev and 3D Graphics - some basic terminology to get you started

14 Coordinate Systems - how Virtools Dev measures things, controls position and controls orientation; the basics of the mathematics behind the scenes

15 Transformations - how to translate, rotate, and scale with a minimum of mathematics

16 Matrix Operations - a brief introduction to the mathematics behind transformations

17 Worlds and Levels, Places and Scenes - how Virtools Dev implements classic game constructs

18 Cameras and Rendering - how Virtools Dev decides what to show in the render window

19 The Render Engine - an overview of the Virtools Dev rendering architecture

13 VIRTOOLS DEV AND 3D GRAPHICS

With Virtools Dev, you can create a virtual world called a *composition* (CMO). A composition is an arrangement of one or more elements and their associated behaviors.

The term *element* is used as a generic label for any "thing" within a composition that is not a behavior (not a Building Block (BB), Behavior Graph (BG), or Script). For example, elements are often items that have a physical presence in a composition; an element can often be seen when a composition is played (e.g. a character). However, elements can also be abstract and used to define logical associations (e.g. members of a group, an array) or data sets (e.g. the data points that define the vertices used to render a character, the description of an animation sequence).

A *behavior* is a description of how an element responds to some form of input stimulus. Applying a behavior to an element makes the element interactive, either with

- the User, or
- other elements of the composition.

You are invited to review the materials contained in the Virtools Dev Glossary and to refer to the Glossary any time you see a term that has a specific meaning within Virtools Dev.

To be effective at creating real-time, interactive 3D content with Virtools Dev, you should understand some of the basic principles of 3D graphics and how Virtools Dev implements them.

14 COORDINATE SYSTEMS

Coordinate systems define the space or volume of a simulated environment (called a *Level* in Virtools Dev, often called a *World* in other products).

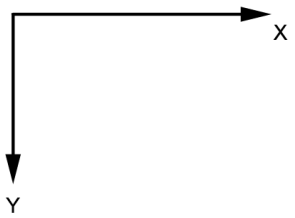
Virtools Dev uses two coordinate systems: a 2D coordinate system for screen positions and a 3D coordinate system for rendering 3D elements.

14.1 The 2D Coordinate System

2D coordinate systems are defined by two axes, at right angles to each other, that define a plane (called the XY plane). The X axis is horizontal, the Y axis is vertical, and the point at which the two axes intersect is the origin of the 2D coordinate system.

In Virtools Dev, 2D operations typically affect the placement of some element on the screen (e.g. the position of the player's score). Therefore, 2D coordinates are also known as *screen coordinates*, and the origin for all 2D operations is set to the upper left corner of the screen.

14-1 2D Coordinate Systems

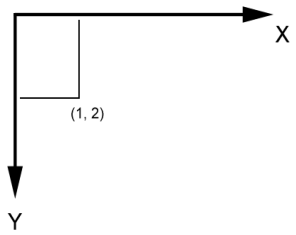


A position on the screen is defined by a *point*. A position is expressed as (X,Y) where:

1. X = the horizontal displacement of the point from the origin, and
2. Y = the vertical displacement of the point from the origin.

For example, **14-2** illustrates a 2D point with coordinates (1,2).

14-2 2D Point



The value of a 2D Coordinate can be expressed in:

- pixels, known as *absolute coordinates*, or
- *normalized* values (from 0.0 to 1.0) that are scaled to the actual screen resolution, known as *homogeneous coordinates* or as *relative coordinates*.

  Homogeneous coordinates offer greater portability for compositions because the screen elements are automatically scaled to match the screen.

14.2 The 3D Coordinate System

The 2D coordinate system introduced above can be extended to three dimensions to take into account not just height and width but also depth.

3D coordinate systems are defined by three axes, where each axis is at right angles to the other two axes. The point at which the three axes intersect is the origin of the 3D coordinate system, also known as the origin of the *world coordinate system*.

Together, the three axes define three planes:

1. the XY plane where Z is a constant value,
2. the XZ plane where Y is a constant value, and
3. the YZ plane where X is a constant value.

NOTE The center of the world coordinate system is not necessarily the center of the visible elements in the Level. You can place the visible elements of your composition at any position that you choose, not just centered about the origin of the world coordinate system.

14.2.1 Units of Measurement

The Virtools Dev default units of measurement are expressed in the metric system. That is, one unit in the default 3D coordinate system corresponds to one meter in the real world. However, there is no enforcement of this relationship within Virtools Dev; you are free to build your models to any scale that you choose.

There are a number of reasons for building your models to true scale:

1. You greatly reduce the probability that your models will require changes within Virtools Dev; the relative sizes of your models will be as expected.
2. The hardware support in video cards is of fixed precision. If your models are created at greatly inflated scale (e.g. a person modeled as 1,000 units tall), then your video card is likely to exhibit rendering artifacts – particularly with respect to depth sorting.
3. There are a number of highly optimized collision detection mechanisms within Virtools Dev that require that obstacles be set to a unit scale factor (scale = 1) for best performance.

   The scale factor is a measure of the current size of the model relative to the size of the model at the time the model was brought into Virtools Dev. Therefore, the scale factor is independent of the actual size of the model, as expressed in World units.

NOTE Virtools Dev provides a mechanism to reset the scale factor of a model to unit scale after resizing but it is easier if the model is built at the correct size from the start.

NOTE There is no mechanism to reset the scale factor of a Character to unit scale.

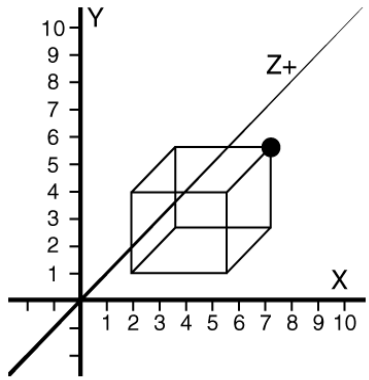
14.2.2 3D Coordinates

A 3D coordinate is a position in a 3D coordinate system (i.e. within a level), expressed as (X,Y,Z) where:

1. X is the horizontal displacement of the point from the origin,
2. Y is the vertical displacement of the point from the origin, and
3. Z is the depth displacement of the point from the origin (the displacement into or out of the screen).

For example, **14-3** illustrates a 3D point with coordinates (4,4,4).

14-3 3D Point



14.2.3 Vectors

Direction (such as the direction a character is facing) and length (such as the distance between two elements in a level) are expressed as *vectors*. A vector is expressed as (X,Y) for vectors within a 2D coordinate system (called a **Vector2D** in Virtools Dev) and expressed as (X,Y, Z) for vectors within a 3D coordinate system (called a **Vector** in Virtools Dev).

NOTE The syntax for expressing points and vectors is identical! Therefore, you must always be careful to determine the context when presented with a value in the form (X, Y, Z).

14.2.4 Local Coordinate System, Relative Coordinate System

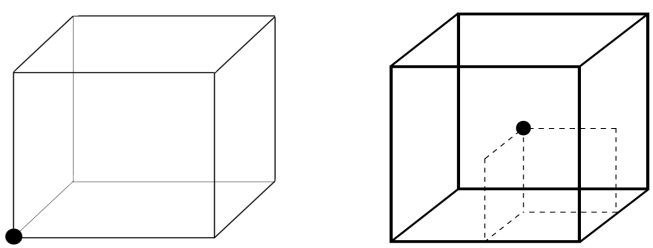
Every 3D Entity has a local coordinate system that is set by the person who created the 3D Entity. The local coordinate system defines the origin for the 3D Entity and the vectors that form the axes of the local coordinate system.

Typical locations for the origin of the local coordinate system are at

1. one corner of the bounding box (the minimum rectangular volume that encloses the 3D Entity), and
2. the geometric center of the 3D Entity (also known as the Barycenter).

Other locations for the origin of the local coordinate system are possible, Virtools Dev places no restrictions on the origin of the local coordinate system.

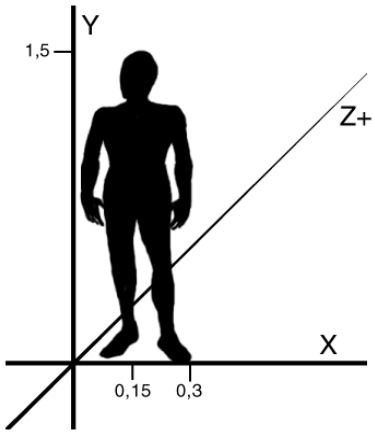
14-4 3D Entity's Bounding Box and Barycenter, with origin of local coordinate system shown



For example, the origin of the character's local coordinate system may be set to the heel of the character's left foot. The positions of the various body parts are then defined relative to the origin of the local coordinate system. The right foot might be 0.3 units from the origin in the positive X direction (+X) and the head might be 0.15 units in the positive X direction (+X) and 1.5 units in

the positive Y direction (+Y).

14-5 Character showing local origin



The position, orientation and scale of an element in the local coordinate system are stored in a data structure called the *local matrix*.

14.2.5 World Coordinate System, Absolute Coordinate System

Virtools Dev also defines a default world coordinate system, a set of axes that are shared by all elements of the composition. The world coordinate system is used as a common reference point for all elements of a composition.

The position, orientation and scale of an element in the world coordinate system are stored in a data structure called the *world matrix*.

Matrix concepts are addressed in greater detail later in "Matrix Operations" on page 71 and "Example Transformations" on page 219.

14.2.6 Referential Axes, Referential

When 3D transformations are applied to an element of a composition, there

must be some point of reference for those transformations. In Virtools Dev, the point of reference is called the *referential*.

The referential is most commonly used as a parameter input (pIn) to a Building Block (BB). The referential is expressed as the name of some element in the composition.

Within Virtools Dev, the world coordinate system is identified by the special term --NULL--. The local coordinate system of any element is identified by the name of the element.

For example, you may want to position an element at (5,0,0) in the world coordinate system. In this case, the referential is the world coordinate system (--NULL--).

Alternatively, you may want to position an element at (5,0,0) relative to that element's current position (independent of where the element is within the Level). In this case, the referential is the name of the element.

Finally, you may want to position a first element at (5,0,0) relative to a second element's current position (independent of where the second element is within the level). In this case, the referential is the name of the second element.

14.2.7 Orientation

Every 3D Entity also has an *orientation*. The orientation of a 3D Entity (including Characters) is defined by three vectors:

1. The *Right* vector.

   From the 3D Entity's perspective, the Right vector points to the right. The default Right vector for a 3D Entity is the positive X axis of the local coordinate system.

2. The *Up* vector.

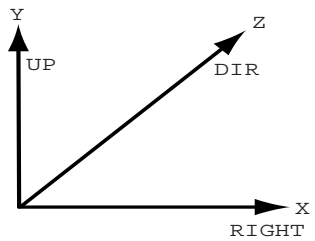
From the 3D Entity's perspective, the Up vector points in the direction that the 3D Entity considers as Up. The default Up vector for a 3D Entity is the positive Y axis of the local coordinate system.

3. The *Dir* vector.

   From the 3D Entity's perspective, the Dir vector points in the direction that the 3D Entity considers as the forward or facing direction. The default Dir vector for a 3D Entity is the positive Z axis of the local coordinate system.

Together, the default values for the Right, Up, and Dir vectors define the Virtools Dev default 3D coordinate system. The association of the Right, Up, and Dir vectors with the X, Y, and Z axes respectively is known as the *default orientation*.

14-6 Default Orientation



NOTE The concept of a *default orientation* for a 3D Entity is a recommendation for ease of use only. You may use any orientation for a 3D Entity.

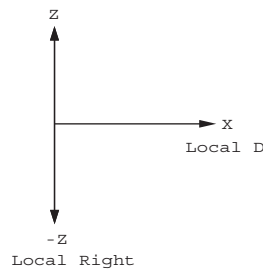
14.2.8 Orientation and the Referential

Orientation is further affected by Virtools Devs use of a Referential for translation and rotation.

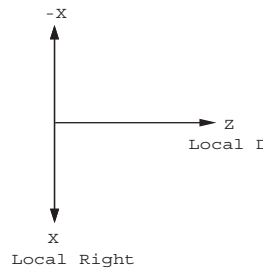
For example, assume that you have two elements in a composition, oriented as shown in **14-7**. Element 1 has a local coordinate system that defines the X-axis as the facing direction (the local **Dir** vector = (1,0,0)). Element 2 has a local coordinate system that defines the Z-axis as the facing direction (the local **Dir**

vector = (0,0,1)). Both elements have the expected **Up** vector of (0,1,0) and both elements face in the direction of the world's X-axis.

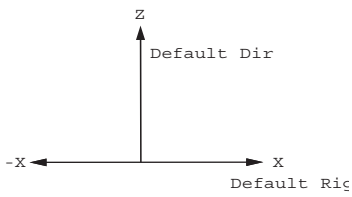
14-7 Orientation and the Referential



Element 1



Element 2



World

NOTE All coordinate systems in **14-7** are viewed from above with the positive Y-axis toward the reader.

Imagine that you are using **Translate** (3D Transformations/Basic) to move Element 1 *forward* five units. You *assume* that this means a translation vector of (0,0,5) because you know that this assumption complies with the Virtools Dev default orientation. What happens when you try to Translate Element 1?

If you try to translate Element 1 with a referential of:

1. --NULL--, then Element 1 moves 5 units in the direction of the world's Z-axis, *independent* of the facing direction of Element 1. Element 1 appears to move to its left.
2. Element 1, then Element 1 moves 5 units in the direction of Element 1's Z-axis, *independent* of the facing direction of Element 1. Element 1 appears to move to its left.
3. Element 2, then Element 1 will move 5 units in the direction of Element 2's Z-axis, *independent* of the facing direction of Element 1. Element 1 appears to move forward, *in the expected direction.*

Had all elements shared a common orientation, the result would have been identical in all cases. There is a sample CMO, Orientation and the Referential.cmo in Documentation\Cmos\FinishedCmos that illustrates this orientation and referential example.

While not strictly necessary, designing your elements to share a common orientation can simplify the task of adding interactivity to a composition.

See "Controlling the Orientation of an Element" on page 217 in the Appendix of this User Guide for further information on controlling the orientation of an element.

NOTE The orientation vectors are not constrained to unit values or a single unit value. For example, the orientation vector (3.5, 5.2, 1.9) is valid.

15 TRANSFORMATIONS

Transformations are simple operations that can be performed on an element:

1. Translation – move the entity.
2. Rotation – change the facing direction of the entity.
3. Scale – change the size of an entity.

You can perform transformations on elements in Author mode by using the transformation tools in **3D Layout**, or changing the parameters for an element in its **Setup**. Alternatively, you can use transformation BBs.

Transformations are always performed relative to a referential entity. The referential entity can be the entity itself, the world coordinate system, or any other entity in the world that has a local coordinate system. **15-1** summarizes the most common transformation BBs.

15-1 Transformation BBs

Translation	
Translate	Move the target entity, relative to the referential entity, within the current frame.
Move To	Move the target entity, relative to referential entity, over a specified number of frames or over a fixed period.
Rotation	
Rotate	Rotate the target entity about the local origin (the origin of the target entity's local coordinate system), according to an axis defined relative to the referential entity coordinate system
Rotate Around	Rotate the target entity about the referential origin (the origin of the referential entity's local coordinate system), according to an axis defined relative to the referential entity coordinate system

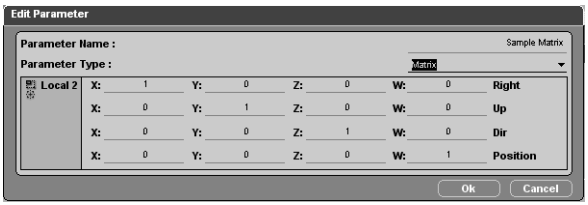
Scale	
Scale	Change the size of the target entity. The new size can specified as Absolute (in the world coordinate system) or as Relative (in the target entity's local coordinate system)

16 MATRIX OPERATIONS

Matrix operations provide the greatest degree of direct control over transformations. However, you do not have to use matrices to manipulate the elements of your composition. Virtools Dev provides many BBs (e.g. **Translate**, **Rotate**, **Scale**, and **Set Position** (all in 3D Transformations)) that allow you to manipulate the elements of your composition without matrices.

Matrices are shown differently in the **Edit Parameters** dialog box, as shown in **16-1**, and in the **Parameter Debugger**, as shown in **16-2**.

16-1 Matrix in Edit Parameters dialog box


Edit Parameter
Parameter Name :
Sample Matrix
Parameter Type :
Matrix
Local 2
X: 1 Y: 0 Z: 0 W: 0 Right
X: 0 Y: 1 Z: 0 W: 0 Up
X: 0 Y: 0 Z: 1 W: 0 Dir
X: 0 Y: 0 Z: 0 W: 1 Position
Ok
Cancel

16-2 Matrix in Parameter Debugger

For greater legibility, in the **Parameter Debugger** the format is

[Right] [Up] [Dir] [Position]

A selection of example transformations in matrix form is provided in the Appendix of this User Guide.

Further details of matrix manipulation are beyond the scope of this User Guide but can be found in most reference materials for Linear Algebra.

17 WORLDS AND LEVELS, PLACES AND SCENES

What is the difference between a World and a Level? What is a Place and a Scene? For many people, their first exposure to these terms is when playing a computer game. You play a game in a World that is divided into Levels – usually because the computers available to the target market for the game do not have the memory capacity to keep the entire World in memory at one time. The Levels in the World are often divided geographically; each Level occurs in a different location within the World. Sometimes, the Levels are mission based; the location may not change but the way the elements within the World act and interact does change.

Virtools Dev provides support for these models, as illustrated in **17-1**.

17-1 Worlds and Levels

Computer Game	Virtools Dev
World	Level
Level (geographically distinct)	Place
Level (mission based)	Scene

The Level is the ancestor element for all parts of a composition. As an element, the Level has the special name --NULL-- when used as a referential.

A Place is an abstract element used to identify the elements found within a physical locale in a composition. For example, within a building, each room can be defined as a separate Place.

A Scene is an abstract element used to define the elements within a narrative unit of a composition and is not restricted to a physical locale within a composition. Scenes can be used to control *which* elements are active *at what time*: in *this* part of the story, only *these* elements are active whereas in *that* part of the story only *those* elements are active.

Places and Scenes are mechanisms used to organize the media within your composition and to optimize the performance of your composition. The elements within a Place or a Scene can be made *Active* or *Inactive*. Only active elements are processed; significant performance gains can be realized through the proper application of Places and Scenes.

18 CAMERAS AND RENDERING

The process of converting a Virtools Dev composition into a screen image is called *rendering*. One or more cameras are used, just like in a movie, to define points of view within a Level.

Each camera has a number of characteristics that can be controlled by the Author, including position, focal length, and field of view. The type of *projection* used by a camera can also be controlled.

A camera using a perspective projection acts like a camera in the real world – images show perspective: objects become smaller with increasing distance and parallel lines converge at the horizon.

A camera using an orthogonal projection does not perform perspective correction – objects remain the same size no matter their distance from the camera and parallel lines remain parallel no matter the distance from the camera. Computer games that scroll from one side of the screen to the other typically use an orthogonal camera.

You might ask why there are two projections. Rendering an image with proper perspective is much more difficult (and requires a lot more processing power) than rendering an image with orthogonal projection.

18.1 Depth of Field, Z buffering

Creating a virtual world that can be rendered in real time is a form of simulation. As just discussed, Virtools Dev uses cameras to control the point of view in a level. One area where the camera simulation is less accurate is depth of field. In a real camera, only certain objects in a scene are sharply focused. The distance at which objects are in perfect focus is called the focal plane and the region about the focal plane where a viewer can not detect any lack of focus is called the depth of field. Objects that are too close to the camera or too far from the camera are out of focus and indistinct.

In a computer simulation of a camera, it is much faster to assume that the camera has a lens with an infinite depth of field so that all objects in a scene, no matter their distance from the camera, are in perfect focus.

If everything is perfectly in focus, how does an individual determine what objects are closer or farther away? Generally, an individual relies upon visual cues (such as the relative sizes of objects or whether one object obstructs the view of other objects) to determine what is near and what is far.

Virtools Dev uses a technique called a depth sort on the rendered objects to determine whether one object should obstruct the view of another object. Given that the Z axis is Virtools Dev's default depth axis, it should not be surprising to learn that the hardware support for depth sorting within a video card is known as the Z Buffer.

In most cases, when rendering 3D Entities you will not be concerned with direct control of the Z Buffer. After all, determining *what* is visible and *where* is one of the principal jobs of the rendering engine.

However, when rendering 2D Entities you will often be concerned with the rendering order. 2D Entities are most often used for fixed visual elements within a composition such as the current score, control panels and menus, and background images. 2D Entities such as the score and menus need to be drawn in front of the 3D elements of the composition. Background images, however, need to seem to be behind all of the 3D Entities *and* behind all of the foreground 2D Entities such as the score. To achieve this effect, the Virtools Dev rendering order is:

1. 2D elements that are defined as being in the background.
2. 3D elements within the view.
3. 2D elements that are defined as being in the foreground.

You can control whether a 2D Entity is rendered in the background or in the foreground. Within the background or the foreground you can also control the order in which 2D Entities are drawn. In all cases, you can easily control

what will be drawn *when* by remembering that smaller numbers (including negative values) mean further away and larger numbers (more positive) mean closer. 2D Entities that are farther away are drawn before 2D Entities that are close.

18-1 Z order number and rendering order, relative to the active Camera

	Closest element		Furthest element
Z order number	32767	0	-32768

19 THE RENDER ENGINE

The Render Engine is the part of Virtools Dev that draws the image you see on-screen. There are actually two components to the Virtools Render Engine, either of which can be replaced or customized using the Virtools Dev SDK:

1. The Render Engine - CK2_3D
2. The Virtools Rasterizers, providing support for DirectX 5, DirectX 7, DirectX 8, DirectX 9 and OpenGL

19.1 The Render Engine – CK2_3D

The Virtools Render Engine is a 3D engine abstraction, as its API is independent of specific 3D hardware and software. This abstraction isolates the rest of Virtools Dev (with the exception of the Rasterizer), and especially the Behavioral Engine, from the details of 3D implementation.

The Render Engine decides what should be drawn, but does not actually draw the image itself. Based on information supplied by the Behavioral Engine (CK2), the Render Engine develops a list of all items that can be seen.

The list of visible items is then provided to the Rasterizer for drawing on-screen.

19.2 Virtools Dev Rasterizers

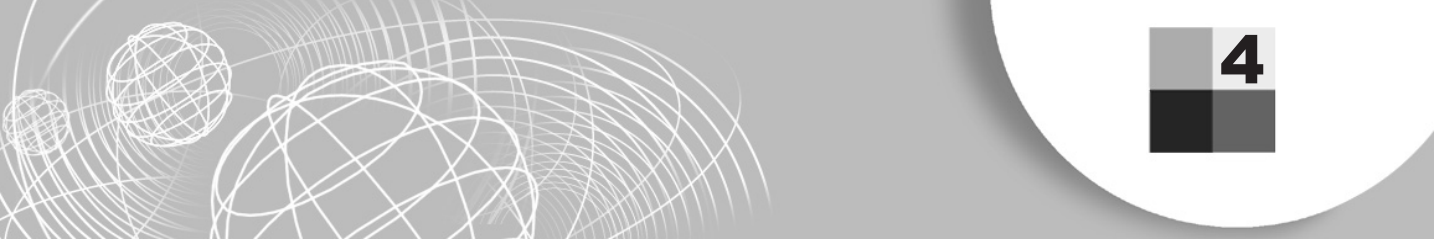
The Rasterizer defines an Application Programming Interface (API) that abstracts the rendering hardware and associated APIs.

The Rasterizer that you or the User can use depends on the capabilities of the graphics card being used.

The Rasterizer deals with polygons, textures and rendering settings. Given these 3D elements, the Rasterizer draws one 2D image intended to be shown on screen. Thus, the Rasterizer implements the performance critical last phase

of the rendering process.

Virtools Dev implements a Rasterizer for five different 3D APIs: DirectX5, DirectX7, DirectX8, DirectX9 and OpenGL.



Part 4 - Understanding Virtools Dev

Understanding Virtools Dev explains (almost) all you ever wanted to know about Virtools Dev but didn't know who to ask. This part answers questions like: *Is it possible to have more than one level in a composition?* and *Why can some behaviors only be applied to certain elements?*

Whatever your experience level, this part is essential reading for everyone. You could get along working in Virtools Dev without reading this part, but why make things hard for yourself?

20 Elements, Classes, and Object Oriented Design - explores and explains the relationships between the elements of a composition

21 The Elements of a Composition (CMO) - the pieces that you assemble to form a composition

22 The Virtools Dev Process Loop - an introduction to processing behaviors and rendering the results

23 The Behavioral Engine - how the Behavioral Engine decides what to do, and when

24 Behaviors and Scripts - the Virtools Dev approach to programming and program flow control

25 Parameters - script variables in Virtools Dev, there is more here than meets the eye at first glance

26 Parameter Operations (paramOps) - an introduction to the data manipulation tools

27 Attributes - element variables in Virtools Dev

20 ELEMENTS, CLASSES, AND OBJECT ORIENTED DESIGN

Virtools Dev organizes *elements* into *classes*. Elements include the media you bring into Virtools Dev (models, sounds, animations, etc.) and the things you create in Virtools Dev (such as Curves and Scenes).

Essential topics in this section include:

- How Virtools Dev classifies media.

  For example, there is a Light class of objects in Virtools Dev to which all lights belong. Each light is an instance of the Light class, and has characteristics such as its color and its range.
- Why certain operations are only possible on certain elements (classes).

  For example, only characters can use character animations.
- Why other operations can be applied to all elements.

  For example, all elements can be activated or deactivated.

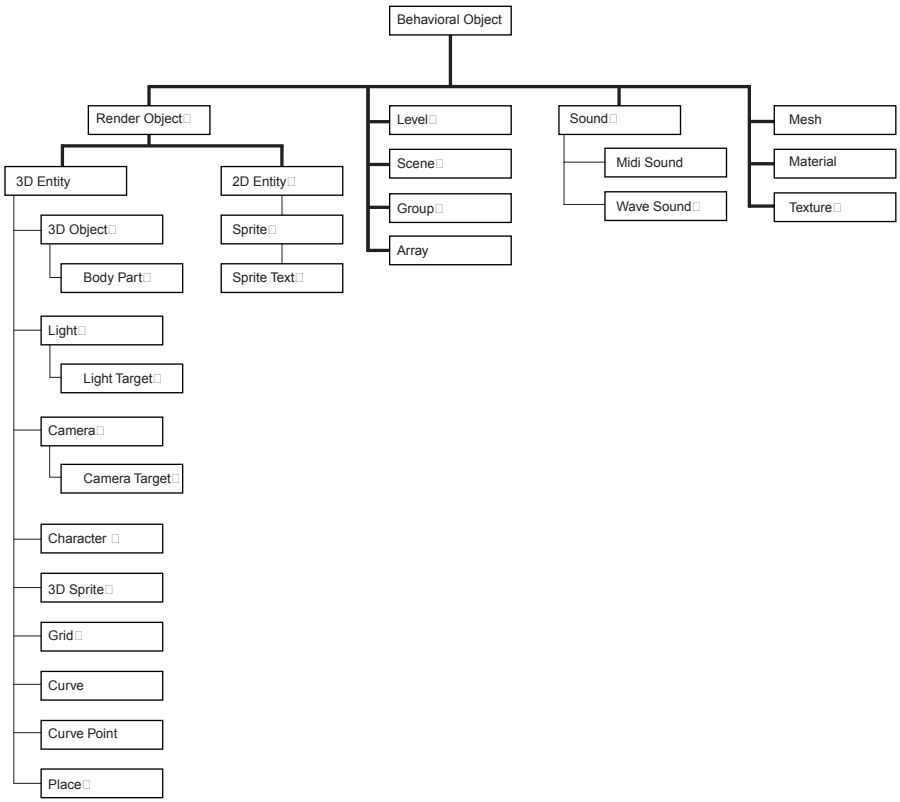
This section presents a simplified version of the Virtools Dev class system. You can find a more detailed description of the Virtools Dev class system in the SDK documentation.

20.1 Object Oriented Design

Virtools Dev takes an Object Oriented approach to building a composition (a CMO file). That is, every element is an instance of a unique class where each unique class is described by a *class definition*. The term *CKClass* is used as a generic label for any class or class definition used by Virtools Dev. **20-1** illustrates the Virtools Dev class hierarchy.

Elements are controlled using *methods* encapsulated within behaviors and parameter operations.

20-1 Simplified Class Hierarchy



For example, there is a CKClass called CKCharacter. Only CKCharacters can use the behaviors specifically designed for characters (such as the Character Controller).

The object oriented nature of Virtools Dev means that everything you import into Virtools Dev (the elements in the form of models, sounds, etc.) becomes an instance of a CKClass. Also, anything that you create in Virtools Dev (from a camera to an array) becomes an instance of a CKClass as well.

20.2 Inheritance

One advantage of using a class hierarchy is the principle of *inheritance*. That is, any element has its own unique characteristics *and* the element inherits the characteristics of all of its ancestor classes.

For example, the class CKLight descends from

1. CK3DEntity, which descends from
2. CKRenderObject, which descends from
3. CKBeObject.

Therefore, a light has special characteristics that apply only to objects of the CKLight class – characteristics like:

- the type of light (point, directional, etc.)
- the color of the light (white, red, etc.)
- the range over which the light is visible

These characteristics can be changed via the light's setup or via a behavior.

Secondly, a light is a 3D Entity, which means it has inherited the characteristics of a 3D Entity– characteristics like position and orientation in the 3D space. Since a light is also a 3D Entity, any behavior that can be applied to a 3D Entity can also be applied to a light.

Thirdly, a light is a Render Object, which means that it can be rendered (seen in playback). Since a light is a Render Object, any behavior that can be applied to a Render Object can be applied to a light.

Finally, a light is a Behavioral Object, which means that it can have a behavior attached to it (some Virtools Dev classes, such as classes that store internal data only, can not have behaviors attached to them). Since a light is a Behavioral Object, any behavior that can be applied to a Behavior Object can be applied to a light.

For more information on the specific properties of a class, see the Setup for

that element in the Online Reference.

20.3 Specialization

Virtools Dev supports the specialization of inherited behaviors so that the behaviors can be optimized for a specific task (also known as *polymorphism*). For example, moving a 3D Frame is much simpler than moving a Character so the behavior for moving a 3D Frame can be optimized for the simpler task.

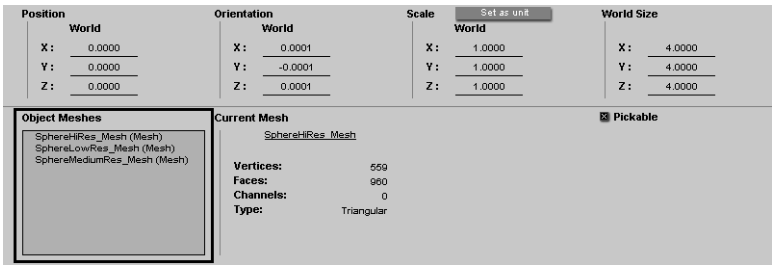
Optimized behaviors reduce the computing time for a given task resulting in a composition that is smaller, that is more responsive to user input, and that renders at an acceptable frame rate.

20.4 Aggregation

Virtools Dev supports aggregation, the logical relationship between elements where a first element is a part of a second element yet both elements are distinct.

For example, if you look at the **3D Object Setup** in **20-2**, you can see a column labeled **Object Meshes**. A 3D object may have several meshes, although only one mesh may be active at a time.

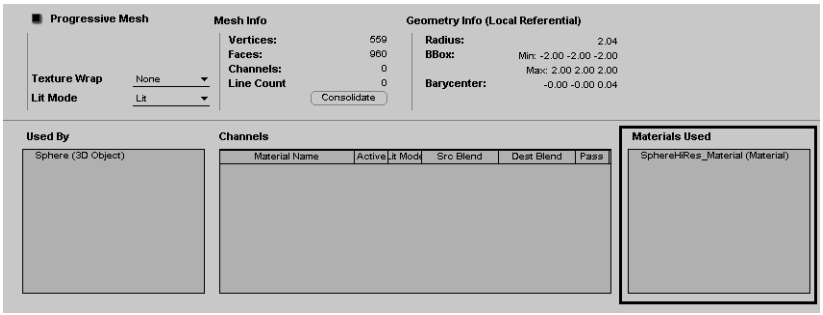
20-2 3D Object Setup with Mesh column shown



If you look at the **Mesh Setup** in **20-3**, you can see a column labeled **Materi-**

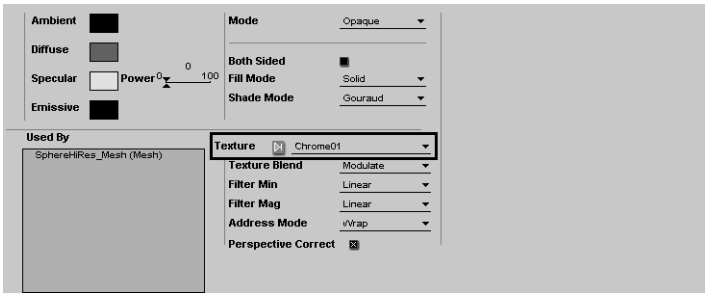
als Used. A mesh may have several materials, several of which may be active at a time.

20-3 Mesh Setup with Materials column shown



If you look at the **Material Setup** in **20-4**, you can see a field labeled **Texture**. A material may have only one texture.

20-4 Material Setup with Texture shown



Therefore, the elements that form a 3D entity (whether it be an object, a character, etc.) are related, yet each element remains independent. In this example, a Texture is part of a Material, a Material is part of a Mesh, and a Mesh is part of a 3D Object.

Because each element remains independent, the characteristics of each element (in this example, the object's mesh, material, and texture) can be changed

quickly and easily. In fact, the entire element can be exchanged with another compatible element! For example, you could change a 3D Object's Mesh, Material, or Texture – or any combination of these without changing the fact that the 3D Object exists.

NOTE If you change the mesh on a Character, it is very likely that your Animations will no longer work as expected.

20.4.1 Run-time Aggregation, The Scene Hierarchy

A Scene is the set of elements that are currently active at runtime. The elements within a Scene are organized into a scene hierarchy.

Virtools Dev supports a special form of aggregation within the scene hierarchy. At runtime, relationships between elements can be established via **Set Parent** and **Add Child** (both in 3D Transformations/Basic). Run-time aggregation allows you to establish relationships between an arbitrary set of 3D Entities, typically for the purpose of simplifying the application of transformations to the set of 3D Entities.

For example, even though it would be easier to do in a modeling application, using **Set Parent** and **Add Child** you could construct a car as a hierarchy of 3D Entities: a car has doors, a body, and wheels. Once the hierarchical relationship is established, transformations on the car automatically transform the children of the car: the doors, body, and wheels.

NOTE **Set Parent** and **Add Child** establish relationships within the scene hierarchy *only* and not within the CKClass hierarchy. These relationships remain after a composition stops playing unless Initial Conditions (ICs) are set on the affected elements *before* **Set Parent** or **Add Child** are activated.

20.4.2 Sharing Elements

Virtools Dev's support for aggregation allows you to share elements such as sounds, animations, meshes, materials and textures throughout your composition.

For example, two chairs could share the same mesh, material, and texture – the chairs would appear to be identical and would differ only in name. However, the chairs could have the same mesh, but have different materials and textures – the chairs would then have the same form, but look different.

Sharing elements can greatly reduce file size and the work-load for CPUs and graphic cards. Reducing the number of elements in a composition also makes managing the development of your composition easier.

20.5 Association

Association allows Virtools Dev to create a logical relationship between elements that are not directly related through a parent-child inheritance relationship in the CKClass hierarchy. Objects that are associated communicate with each other but remain distinct in all respects.

For example, a character is often associated with a Group of 3D Entities for the purpose of collision detection, management, and prevention.

21 THE ELEMENTS OF A COMPOSITION (CMO)

21-1 shows the Virtools Dev class hierarchy slightly differently than in **20-1** on page 84. Comments have been included to help you place each CKClass in context with respect to the other CKClasses. Indentation indicates that the indented label is a child of the closest previous level of indentation. For example, a 2D Entity and a 3D Entity are children of RenderObject. Sprite Text is a not a child of RenderObject but Sprite Text is a descendent of RenderObject.

You can find a complete description of Virtools Dev classes in the Online Reference.

21-1 Composition (CMO) elements, according to type and arranged alphabetically

BeObject - an Object to which a behavior can be applied
- RenderObject - an object that is rendered (can be seen in Play mode)
  - 2D Entity - an object that has width and height but no depth
    - Sprite Text - a 2D Entity used to place text in the render window
  - 3D Entity - an object that has width and height and depth
    - Character – a type of 3D Entity that acts as an intelligent entity, directed by the user or by software
- Objects that may be visible but usually are not
  - Camera – an object that defines a point of view
  - Curve - a set of 3D Entities that define a curve in 3D space
  - Grid – a 2D data set whose value depends on 3D coordinates
  - Light – an object that provides illumination
- Objects that are not visible (abstract, data sets, associations)
  - Array – a data set expressed as a table

Group – an arbitrary collection of elements with no restriction as to type
Level – the parent object of the entire composition
Material – the surface characteristics of a mesh
Mesh - the set of vertices that define the shape of an object
Place – a collection of geographically related objects
Scene – a collection of temporally related objects
Sound – a sound stored in digital form
Texture – an image used to provide fine detail on the surface of an object

21.1 The Behavioral Object (BeObject)

While behaviors are the central concept in Virtools Dev, they are of no use if they are not attached to elements. Note that all entries in the simplified view of the Virtools Dev class hierarchy descend from the Behavioral Object (CKBeObject) class. Objects instantiated from these classes (descendants of CKBeObject) are the objects that *can* have behaviors attached to them, but they are *not required* to have behaviors.

When a BeObject has behaviors attached, the BeObject is said to be the *owner* of these behaviors.

NOTE The owner of a behavior is not always the target of the behavior. The behaviors attached to a BeObject may affect the owner or they may affect other objects.

21.2 The Level

A Level can be described as the

- global container for all objects of the composition
- root of all elements in a composition
- ancestor or parent of all elements in a composition

There is only one Level in a composition. The Level can contain one or more Scenes and zero or several Places to organize your composition into manageable portions.

NOTE The Level always contains at least one scene: the Level Scene.

A Level is a BeObject. Therefore, behaviors and attributes can be attached to the Level. Level scripts are typically used to manage Scenes, Places and other elements needed across several scenes (such as Arrays).

NOTE Scripts attached to the Level are automatically attached to all Scenes.

21.3 Scenes

A Scene in Virtools Dev is just like a scene in the movies:

- only one Scene is filmed (active) at a time - the director can only be in one place at a time!
- only the actors (elements in Virtools) in that scene are active - everyone else must wait their turn
- only those active in the scene are filmed by the camera (in Virtools, only active elements are processed by the Behavioral Engine and rendered)

The elements that are not members of the active scene are, by definition, inactive. They are not rendered and their scripts are not processed.

Therefore, scenes are the perfect way to segment compositions into large scale narrative elements or large time-based periods. Scenes are very useful when working in collaboration with other people: each person can work on a specific scene, which can then be saved as a CMO and later merged into one CMO via **Merge Composition as New Scene** (**File** menu).

The Level Scene (containing all the objects of the Level) is made the active scene of the Level at start-up. When a new scene is made active (started), the previously active scene is automatically deactivated.

A Scene is a BeObject. Therefore, behaviors and attributes can be attached to the Scene.

Scenes contain references to elements, not copies of elements or the elements themselves.

21.4 Places and Portals

Just as Scenes are a way to organize time segmentation within a Level, Places are a way to organize space segmentation in a Level or Scene. There may be zero or more places in a Level. There is no specific link between Scenes and Places.

Places are part of the 3D Entity hierarchy and may contain zero or more 3D Entities. Each Place is deemed the Parent of all 3D Entities within the Place.

Places and Portals are used to optimize the rendering phase. The decision of what to display is done relative to the active camera. When the Portal system is active, via Portal Management, the rules are as follows:

1. Before each rendering process, the list of Places spatially containing the active camera (including the view frustum) is computed using the Places bounding boxes.

   These Places, and all the Places that have a Portal with these Places (and all the Places that have Portals with these Places, etc. (recursively)), are considered to be potentially visible and processed by the renderer.
2. If an element is not a member of one of these places, the element is not rendered. The element is immediately considered "out-of-view" (even though the element may be active and its behaviors may be executed).
3. If the Portal system is not active, there is no optimization of the rendering by Places and Portals (every 3D Entity is considered potentially visible and only a hierarchical culling of the scene hierarchy to deter-

mine the potentially visible set of elements is performed), but Places may still be used to hierarchically organize 3D Entities for other purposes.

Places are also very useful when working in collaboration with other people: each person can work on a specific Place, which can then be saved in a Virtools Object File (NMO) and added to a CMO.

A Place is a BeObject. Therefore, behaviors and attributes can be attached to the Place.

21.5 Abstract Elements

Abstract Elements are data structures used by Virtools Dev to store information and to describe relationships between elements.

21.5.1 Groups

Groups are the simplest yet most powerful organizational entity of Virtools Dev. Groups are ordered lists of Behavioral Objects that can implement any logical association that may be required and are not restricted to containing elements of the same CKClass or type.

Groups can be created in Author mode or by BBs at run-time. Groups can be reordered, iterated over, etc. – many standard BBs are provided for group management.

A Group is a BeObject. Therefore, behaviors and attributes can be attached to the Group. Groups can be part of other Groups.

21.5.2 Arrays

Arrays are simple tables containing cells organized as rows and columns. Arrays are organized as typed columns (all the elements of a column must have the same type).

Arrays support five data types:

1. Integer
2. Floating Point
3. String
4. CKObject
5. Parameters

Arrays can be created in Author mode or by BBs at run-time. Arrays can be sorted, filtered, searched, iterated over, etc. - many standard BBs are provided for array management.

An Array is a BeObject. Therefore, behaviors and attributes can be attached to Arrays.

NOTE The value of a parameter in a script can be stored in an Array via a Parameter Shortcut. See the Online Reference (Array Setup) for further details.

22 THE VIRTOOLS DEV PROCESS LOOP

The Virtools Dev process loop, often referred to as a Frame or a Rendering Frame, is a repetitive process that occurs when you (or a User) play a composition. Understanding the process loop will help you to create efficient scripts and smooth running compositions with a satisfactory frame rate.

A satisfactory frame rate is usually defined as a frame rate that supports effective real-time visualization. Real-time visualization starts with a minimum of 15 images per second, with immersion beginning at 60 frames a second. Therefore, as an author, you should always attempt to maintain a frame rate of at least 60 frames per second with a goal of 30 frames per second or more (approximately the same rate as television).

NOTE Virtools Dev has more processing to perform than a standard Player. Therefore, frame rates in Virtools Dev will be slightly lower than the frame rates in a standard Player

Virtools Dev is a real-time engine: it allows any behavior to react constantly and consistently to its environment, including the User.

When a composition plays, Virtools Dev repeatedly performs the Process Loop - always performing the same steps in the same order until a composition is stopped or reset.

22-1 Process Loop

1 Frame		1 Frame		And so on…
Process Behaviors	Rendering	Process Behaviors	Rendering	

As shown in **22-1**, there are two parts to the process loop:

1. Processing Behaviors, and

2. Rendering the Scene.

The duration of one process loop is commonly called a Frame. The frame rate is the number of times the process loop is performed each second and is measured in Frames Per Second (FPS). The frame rate is constantly displayed in the Virtools Dev interface, adjacent to the Play/Pause button at the bottom right corner of the screen.

22-2 Bottom right region of Virtools Dev - FPS display is just to right of Profiler

The process loop should execute as quickly as possible:

- A high frame rate provides rapid response to the User's input, increasing the user's perception of the quality of the composition.
- Rendering the image as often as possible increases the perceived quality of animations and visual effects. Note that rendering the image faster than the refresh rate of the display does not enhance the quality of the visuals and may, in some cases, actually degrade the quality of the visuals.

To maximize the frame rate, you must minimize the amount of behavior processing performed in each frame. Never try to do more in a given frame than the minimum necessary to maintain the immersive experience. When constructing scripts, you should try to:

- divide tasks into sub-tasks and only process the minimum number of sub-tasks per frame.
- structure your scripts such that the frame rate stays relatively constant.
- avoid scenarios where, in response to user input, a large script must be processed in its entirety in a given frame. The frame rate is likely to noticeably drop and the user will be reluctant to provide that same input again.

Some BBs, such as **Character Curve Follow** (Characters/Movement), are

designed to operate in an incremental manner. Rather than following the designated Curve from the starting point to the ending point in a single frame, **Character Curve Follow** (Characters/Movement) moves the Character along only a small portion of the Curve in a given frame – spreading the work across many frames and increasing the quality of the immersive experience.

22.1 Processing Behaviors

Behavioral processing creates the interactivity within your composition. Behaviors can be applied to almost every element in Virtools. All active behaviors are executed during behavioral processing, one after the other, using a sophisticated prioritizing scheme (see "The Behavioral Engine" on page 101 for further information).

Each behavior, when executed, may activate other behaviors through behavior links. Behavior links have a link delay and the link delay is measured in Frames (process loops). The Link Delay can be:

- 0 – propagate the activation within the current frame
- 1 – propagate the activation in the next frame
- n - propagate the activation in the n^{th} frame after the current frame

Various managers help the Behavioral Engine in its work. The managers perform their tasks either at the start of behavioral processing or at the end of behavioral processing (just before rendering).

In general, you do not need to know how or when these managers work, with one exception: the MessageManager. Messages are sent at the end of behavioral processing and are received in the following frame. Therefore, interactions driven by messages *always* experience a one frame delay between the frame in which the message is sent and the frame in which the message is received.

A complete description of all Virtools managers is available in the Virtools SDK.

22.2 Rendering

Rendering displays the composition and is performed by a separate render engine. The engine is chosen by the User and depends on the capabilities of their graphics card and operating system.

You can choose the render engine (**Render Device**) for Author and Play modes while working in Virtools Dev: from the **Options** menu choose **General Preferences** and select the **3D Layout - Rendering** tab.

Typically, rendering is the most time intensive portion of the process loop and is highly dependent on the capabilities of the underlying hardware.

23 THE BEHAVIORAL ENGINE

The Behavioral Engine implements behavioral processing and is the central component of Virtools technologies. Also known as CK2 (hence the CK references you often see in the interface, such as CKID, CKClass, etc.), the Behavioral Engine is what makes Virtools technologies so flexible.

What does CK2 do? CK2 executes your compositions, processing scenes and managing all interactions between the User and the elements of your composition. CK2 also processes the elements of your composition that have behaviors applied to them – that is, the Behavioral Objects.

Once all behaviors have been processed for a given frame, CK2 provides the appropriate information to the rendering engine so that the results of the interactivity can be displayed to the User.

When a composition is played, a series of tasks is performed, as outlined in **23-1**.

NOTE To properly interpret the following series of tasks (the algorithm) you must take note of the indentation level of each statement. An indentation indicates a subtlest that must be performed as part of the prior sequence of tasks. For example, task 3.1.1 states "For each BB attached to Start that has not yet been activated, activate the BB with the highest priority". This means that all subtasks (3.1.1.1, 3.1.1.2 *and* their subtasks) must be performed to complete task 3.1.1. The "For each" portion of the task description means that this task (and all subtasks) is performed repeatedly until the required condition – that all BBs attached to Start have been activated – is met.

23-1 Behavioral Process Loop (Frame)

1. For each active element within a Level, sort the elements in priority order to establish the element processing order.
2. For each element, the scripts applied to that element are sorted in priority order to establish the script processing order for that element
3. For each element, from the highest priority to the lowest priority, process the highest priority element remaining
    3.1. For each script attached to the currently processing element, process the highest priority script remaining
        3.1.1.For each BB attached to Start that has not yet been activated, activate the BB with the highest priority
            3.1.1.1 Process the BB, calculating pOuts, activating any bOuts, and sending any messages
            3.1.1.2 For each bOut that activates a behavior link
                3.1.1.2.1. If the behavior link has a delay of 0 then activate and process the next BB. Continue in this manner until all BBs in this part of the Script are processed or the behavior link has a delay greater than 0.
                3.1.1.2.2 If the behavior link has a delay greater than 0 then add the BB to the stack for processing in the required frame. For example, if the link delay is 1, then the BB is processed in the next frame. If the link delay is n, then the script is kept active and the BB is processed in the n^{th} frame after the current frame.

NOTE The Level is always active and is the highest priority element in every frame. All currently inactive elements are ignored in the remaining tasks.

Processing continues in the current frame until all BBs are processed or the link delay is greater than 0. When all behavioral processing is complete, CK2 provides the required information to the rendering engine to draw an image

on screen, the image is rendered, any link delays that are greater than 0 are reduced by 1, and behavioral processing starts again.

You can watch behavioral processing in action by activating Trace mode in the Schematic. When the composition is played, the sequence of activations and BB processing are highlighted in red in the Schematic.

23.1 Behavior Loops

A behavior loop is formed when a series of BBs (and/or BGs) are connected by behavior links (bLinks) in a manner that causes the BBs to be activated or triggered in a repetitive fashion.

In other words, a behavior loop is the visual representation of a repetitive operation. A repetitive operation is also known as an iteration and the process of repeating an operation is also known as iterating.

In any implementation of a repetitive operation, a significant concern is to ensure that there is an acceptable upper limit to the number of times that the operation repeats. In other words, there needs to be a mechanism to ensure that an operation is not repeated indefinitely.

Virtools Dev allows the you to define the maximum number of operations that may be performed in a single Frame via the *Max Behavioral Iterations* setting in the Schematic.

NOTE The default value for Max Behavioral Iterations is 8000. The value of Max Behavioral Iterations is saved with the composition.

Any behavior loop may have a cumulative delay of zero, as long as the loop is not constantly active and as long as the number of iterations in a frame does not exceed the value of Max Behavioral Iterations. If the number of iterations in a frame exceeds the value of Max Behavioral Iterations then the behavioral engine assumes that an infinite loop has occurred and processing is halted.

For example, Array and Group iterators (Logics/Array and Logics/Groups) often have a loop delay of 0 so that all elements within the Array or Group are

operated on within a single frame.

However, always keep in mind that (to maintain a consistent frame rate) you want to divide the processing to be performed into the minimum amount necessary per frame. Therefore, unless absolutely necessary, a behavior loop should have a cumulative delay of at least one frame. The actual behavior loop delay can be any positive value in the range of 0 to 32767.

NOTE Looping BBs (such as **Timer** (Logics/Loops)) that are time-based require a cumulative loop delay of 1 frame or the BB automatically changes to frame-based processing.

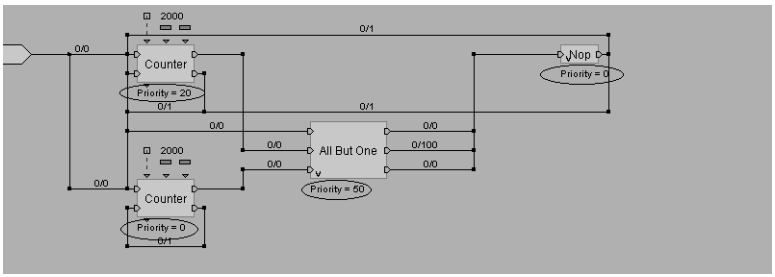
23.2 Priority

Once CK2 has sorted which elements are active, how does CK2 know which element to process first? The answer lies with that element's priority: the highest priority element is processed first.

You can set the priorities for Behavioral Objects, Scripts and Building Blocks. The priority of a Behavioral Object and the priority of a Script are set in the Level Manager. The priority of a BB is set in the Schematic.

If two or more elements have the same priority, then the processing order is random.

23-2 Priority



With respect to **23-2**, note that both **Counters** (Logics/Loop) and **All But**

One (Logics/Streaming) are all connected to Start with a 0 frame delay. Therefore, the Behavioral Engine must determine which BB to activate first based on the priority of each BB.

23-3 Priorities as seen in **23-2**

BB Name	Processing Priority
All But One	50
Counter (at top of **23-2**)	20
Counter (at bottom of **23-2**)	0

The processing order for the first frame is as follows.

1. **All But One** (Logics/Streaming) at priority 50: the top bIn (In 0) is activated by Start then the lower two bOuts (Out 1 and Out 2) are activated
2. The top **Counter** (at priority 20): counts 2000 times then activates **All But One** (following a path with a 0 frame delay bLink)
3. **All But One** (at priority 50): the middle bIn (In 1) is activated by Start then the outer two bOuts (Out 0 and Out 2) are activated
4. The bottom **Counter** (at priority 0): counts 2000 times then activates **All But One** (following a path with a 0 frame delay bLink)
5. **All But One** (at priority 50): the lowest bIn (In 2) is activated by Start then the upper two bOuts (Out 0 and Out 1) are activated

At the end of the first frame **All But One** has been activated three times – probably not the expected result.

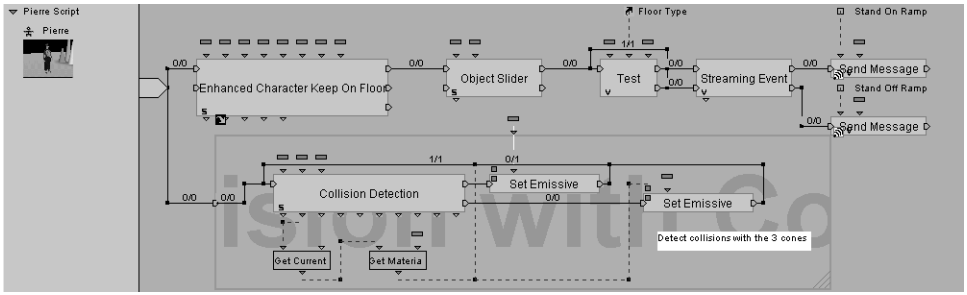
NOTE As a general rule, when multiple BBs can activate the same BB (as in this example), the receiving BB should have a lower priority than the BBs that can activate it to ensure that all prior BBs are processed first.

24 BEHAVIORS AND SCRIPTS

A Behavior is expressed as a script - the visual representation of a behavior, applied to an element, as represented in the Schematic.

A script is composed of two parts – a header and a body. The script header displays the name of the script and the owner of the script. The script body is composed of the Start and one or more BBs, BGs, paramOps, Parameters, bLinks, pLinks, comments, etc.

24-1 Sample Script



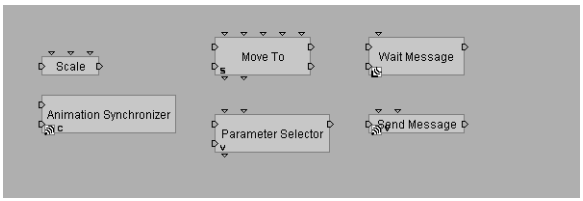
24.1 Behavior Building Block (BB)

The fundamental mechanism used to implement behaviors is the BB. BBs are a visual representation of a software element known as a function, a ready-to-use solution to a known task.

24.1.1 Interpreting a BB Symbol

BBs come in various rectangular sizes. You can see some different types of BBs in **24-2**.

24-2 Different types of Building Blocks



24.1.2 Behavior Input, bIn

A BB typically has at least one Behavior Input (bIn) – although there are a few exceptions (BBs that operate in Author mode, e.g. **Create Blended Animation** (Characters/Animation)). bIns are always located on the left side of the BB. A BB starts processing when the BB receives an activation at a bIn.

24-3 Behavior Inputs (bIns)

24.1.3 Behavior Output, bOut

A BB often has at least one Behavior Output (bOut) – although there are numerous exceptions. bOuts are always located on the right side of the BB. bOuts are activated at the end of processing within the current frame. Activation flow follows any attached Behavior Links.

24-4 Behavior Outputs (bOuts)

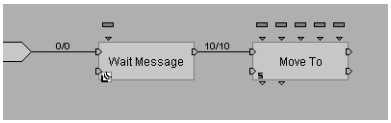
24.1.4 Behavior Link, bLink

BBs are connected by Behavior Links (bLink) that define the order in which

BBs are processed. The sequence of BB processing is called the activation flow whereby activations propagate across bLinks.

Every bLink has a Link Delay that specifies *when* CK2 should process the BB at the end of the bLink. The Link Delay can be 0 (meaning that the BB is processed in the current frame) or *n (*meaning that the BB is processed in the *nth* frame after the current frame).

24-5 Two Behavior Links (bLinks) - left link has link delay of 0, right link 10



24.1.5 Parameter Input and Parameter Output

BBs typically have Parameters, either in the form of Parameter Inputs (pIns) above the BB that receive data or Parameter Outputs (pOuts) below the BB that transmit data, as you can see in **24-6**. Parameters are discussed in more detail in "Parameters" on page 117.

24-6 Parameter Inputs (pIns) and Parameter Outputs (pOuts)

24.1.6 Target Parameter

A Target Parameter is a special type of pIn used to explicitly identify the element affected by the BB.

When a BB is attached to an element, that element becomes the *owner* of the Behavior. Typically, a BB attached to an element is *implicitly targeted* at the owner of the Script.

For example, **Translate** (3D Transformations/Basic) normally modifies the

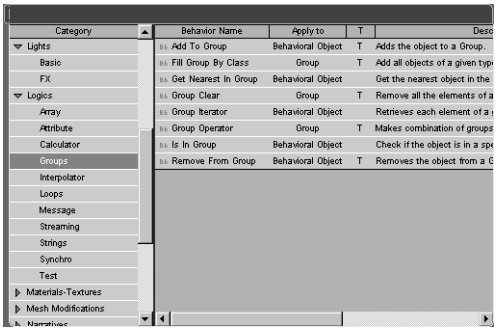
position of its owner.

However, you may want a BB to affect a different element than the owner. In this case, you must *explicitly target* the BB at a different element.

Alternatively, you may attach a Behavior to an element of a different type than the type supported by the Behavior (for example, you can attach a **Rotate** (3D Transformations/Basic) behavior to a texture). A Target Parameter is automatically generated (for a targetable BB) by Virtools Dev in the case of an incompatible class.

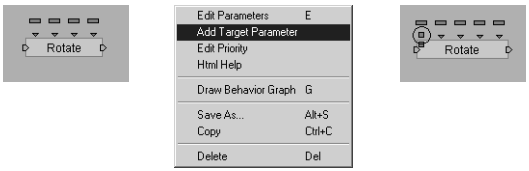
A BB is only *targetable* if there is a "T" in the targetable column (between the "Apply to" and "Description" columns) within the Building Blocks window.

24-7 Targetable column of Building Blocks window



If a Target Parameter does not already exist on a targetable behavior, a Target Parameter can be added by selecting "Add Target Parameter" from the context menu. A new pIn is created as the left-most pIn. A Target Parameter input is identified by a pair of small squares (rather than the usual single small triangle for a regular pIn).

24-8 BB without Target Parameter then with Target Parameter



24.1.7 C, S, and V

Some BBs are marked with one or more of the letters C, S or V in the lower left corner.

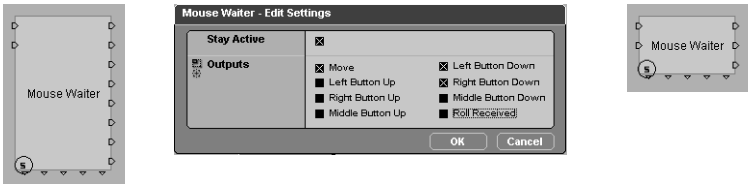
C in the in the lower left corner indicates that the BB has a Custom Dialog Box used to configure complex parameters (e.g. **Animation Synchronizer** (Characters/Animation)).

24-9 Custom Dialog Box for Animation Synchronizer

S in the lower left corner indicates that the BB has Settings. Settings typically control *which* parameters are processed by the BB or *how* the parameters are

processed by the BB.

24-10 After using Edit Settings, only 3 bOuts are calculated



For example, **Mouse Waiter** (Controllers/Mouse) normally calculates 8 bOuts (see **24-10**). You may only require 3 of these bOuts so you can edit the Settings to calculate only the 3 bOuts you require, thus saving valuable processing time.

Or, you may require a BB such as **Linear Progression** (Logics/Loops) to be frame based rather than time based. You can choose frame based processing via the BB's Settings.

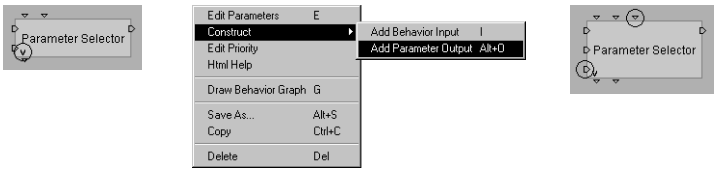
24-11 Edit Settings dialog box for Linear Progression

V in the lower left corner of the BB indicates that the BB has a variable configuration. That is, you can do one or more of the following:

- add bIns
- add bOuts
- add pIns
- add pOut
- change some or all of the types of the pIns and/or pOuts

Check the individual documentation for a BB in the Online Reference to determine exactly what can be changed.

24-12 Variable BB - Parameter Selector



24.1.8 Messages

A BB may have message icons, indicating that the BB sends or receives messages. Typically, messages are used to signal a change in state, to request that some task be performed, and to signal that some task has completed.

NOTE There is a one frame delay between sending and receiving a message. See "The Virtools Dev Process Loop" on page 97 for more information.

24-13 Message sending BB - Send Message

24-14 Message receiving BB - Wait Message

24.1.9 BB Processing

There are three kinds of BB that are processed at run-time, that is, in Play mode.

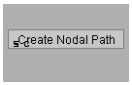
1. Single Action: the BB completes processing within the current frame. A Single Action BB can stand alone or be part of a behavior loop. Example: **Set Fog** (World Environments/Global).

2. Internally Looped: the BB is turned *On* and the BB is activated every frame until the BB is turned *Off*. Example: **Keyboard Controller** (Controllers/Keyboard).
3. Externally Looped: the BB completes one step in the BB's process loop within the current frame. If the Author wants the BB to operate in the typical manner, an external activation feedback loop is required. Example: **Bezier Progression** (Logics/Loops).

NOTE An externally looped BB does *not* require that an external activation feedback loop be present. It is possible to construct a Script in a manner such that the external activation feedback loop is not required.

There are also a few BBs that are not activated at run-time, but when they are attached to an element. An example of such a BB is **Create Nodal Path** (3D Transformations/Nodal Path).

24-15 BB activated when attached to an element - Create Nodal Path



NOTE You should always check the individual Help page from the Online Reference to see precisely how a BB works.

24.2 Behavior Graph (BG)

A Behavior Graph is an author-defined behavior composed of one or more of BBs, Parameter Operations, etc. At first glance, a BG can look almost exactly like a script. However, a BG is distinct from a script because the author of the BG deliberately encapsulated the behavior. BGs encapsulate a behavior in such a manner that the BG can be saved and reused.

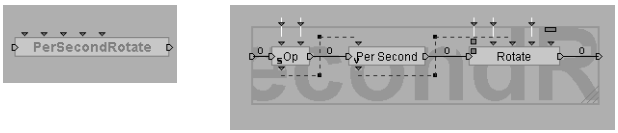
One of the most powerful features of Virtools Dev is this ability to capture interactivity in a reusable form. Behavior reuse can lead to amazing productivity gains!

Virtools Dev treats a BG exactly the same as a BB. A BG can be attached to an

element in the same manner as a BB. A BG can even be considered an Author-defined BB that, to an Author, works in exactly the same way as a BB – a BG can have pIns, pOuts, bIns and bOuts. A BG can be considered an advanced BB constructed to perform a specific task beyond the scope of the BBs provided in Virtools Dev (e.g. a third person camera for a car racing game).

NOTE Any time the Virtools Dev documentation refers to using a (Behavior) Building Block or a BB within a composition, the reader can substitute Behavior Graph or BG.

24-16 Collapsed Behavior Graph (BG) and same BG Expanded



If you compare the image of the closed BG with BB images already presented above, you will notice the differences in the font and border weight used that help to distinguish between a BG and BB.

25 PARAMETERS

A parameter consists of a name (pName), a type (pType), and a value (pValue). Parameters are the Virtools Dev equivalent of variables in traditional programming.

Parameters are used to transfer data across Parameter Links (pLinks):

- between behaviors - from Parameter Outputs (pOuts) to Parameter Inputs (pIns)
- between Parameter Operations (paramOps)
- between paramOps and pIns
- between paramOps and Local Parameters

Parameters can also be used to assign values to an element's Attributes.

25.1 Parameter Types

The type of a parameter defines the kind of data that a parameter can properly hold.

Parameters can be broadly grouped as:

1. References or Pointers

   The parameter identifies an object instantiated from the CKClass hierarchy (e.g. a 3D Entity, an Animation, a Script, etc.).

2. Values

   The parameter contains a data value (e.g. a Color, a Keyboard Key, an Integer, etc.).

3. Enumerations

   The parameter's value is constrained to a range of predefined values (e.g. the Operators supported by **Mini Calculator** (Logics/Calculator) are constrained to +, -, /, *).

4. Special

   The parameter does not fit in the prior categories. There are only three such parameter types, Reflected Object, Obstacle and Floor.

For a more detailed description of parameter types, see the SDK documentation.

25.2 Parameter Input, pIn

pIns are the arguments for Behaviors and paramOps – the values that control how they work. pIns are represented by the small triangles on top of a BB, a BG, or a paramOp as shown in **25-1**. A pIn has a source for its pValue, typically a Local Parameter. pIns do not store their value between activations.

25-1 pIns for BB and for paramOp

25.3 Parameter Output, pOut

When Behaviors or paramOps have finished their processing, they typically produce pOuts. pOuts are represented by the little triangles on the bottom of a BB, BG, or a paramOp. pOuts can have one or more destinations (via pLinks) that are updated as soon as the value of the parameter has changed (the pValue is pushed toward the destination(s)). pOuts store their value between activations.

25-2 pOuts for BB and for paramOp

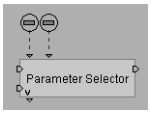
25.4 Parameter Link, pLink

A Parameter Link is a link that propagates a pValue from a pOut or a Local Parameter to a pIn. Unlike Behavior Links, there is no link delay for Parameter Links. Parameter Links are represented in the Schematic by dashed lines.

25.5 Local Parameter

A Local Parameter is a parameter represented by a small rectangle – usually located above a pIn. Local Parameters are data buffers that store values until their value is requested across a parameter link. Local Parameters are normally, but not always, linked to pIns by a pLink.

25-3 Local Parameter attached by Parameter Link to BB



NOTE Two unique Local Parameters are allowed to have the same name. However, this practice is not recommended due to the potential for confusion. For instance, when you copy and paste a Local Parameter, you copy the Parameter's name, the Parameter's type and the Parameter's value to a new local Parameter: these two Parameters are different, even though they have the same name (due to the copy and paste operation). You are encouraged to use unique names for each Parameter that you create.

25.6 This

This is a special local parameter that refers to the owner of the script. When you create a This parameter in a script (via the context menu, see Schematic/Script Body/This Parameter in the Online Reference for further information), it automatically sets its value to the identity of the owner of the script.

25.7 Parameter Shortcuts

Parameter shortcuts are used to:

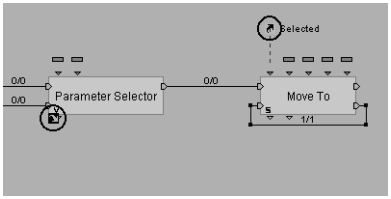
- simplify your scripts by reducing the number of pLinks
- share parameters across script boundaries

Parameter shortcuts are represented by arrow icons, as shown in **25-4**.

Parameter shortcuts are created by copying a pOut (via the context menu) then performing a **Paste as Shortcut** (via the context menu).

Copying and pasting a shortcut creates a second shortcut that has the same source as the initial shortcut. Thus, all copied shortcuts point to the same source and have the same name (pName), type (pType), value (pValue) and identifier (CK_ID) at all times.

25-4 Source (left) and Destination (right) for Parameter Shortcut



26 PARAMETER OPERATIONS (PARAMOPS)

Parameter Operations (paramOps) are simple operations performed on a single parameter or performed between a pair of parameters. A paramOp is represented in the Schematic as a standard sized block with a name, two pIns and one pOut (see **26-1**).

26-1 Three different Parameter Operations (paramOps)

There are three types of paramOp:

1. Data retrieval

   Retrieves information from the Behavioral Engine (e.g. the mouse position)

2. Mathematical operation

   (e.g. multiplication, sine, etc.)

3. Type conversion

   Converts one type of parameter into another type so that the data is in the desired form. For example, a floating point value can be converted to a text string for presentation in 3D Layout.

paramOps can be just as useful as BBs when creating scripts in **Schematic**. Certain operations, particularly some forms of data retrieval, can only be done by performing a paramOp.

NOTE Type conversion can be in the form of a Parameter Operation Link - the default paramOp icon is *not* displayed, the Parameter Operation Link is drawn in a different color than standard pLinks, and the word *Convert* or another conversion type is displayed along the link.

26.1 Parameter Notation

Virtools Dev uses the following notation to describe paramOps in written form.

For binary operations (operations with two pIns), the syntax is

Result or Output **Operation Type** Input1 Input2

where Input1 is the left pIn and Input2 is the right pIn.

For unary operations (operations with one pIn), the syntax is

Result or Output **Operation Type** Input1

where Input1 can be left pIn or the right pIn.

In all cases, the syntax is strictly defined as

<pType> **paramOp** <pType> <pType>

For example:

<Vector> **Multiply** <Float> <Vector>

represents a **Multiply** Operation that produces an output result of pType <Vector>, using inputs of pType <Float> and pType <Vector>.

Values can also be associated with the pIns and pOut.

<Float=50.0> **Multiply** <Float=10.0> <Int=5>

paramOps can be used:

- to perform binary operations

  C=A*B

  V3 = Angle(V1, V2)

  V2=f*V1
- to perform unary operations

  C=sin(A)

C= ABS(A) - get the absolute value of A

- to convert one type of parameter to another

  <Float> **Convert** <Integer>

  <String> **Convert** <Basic Object>

  <3D Entity> **Dynamic Cast** <Character>

NOTE **Dynamic Cast** was implemented to allow any *meaningful* transformation between pTypes that has not been explicitly implemented already as a distinct paramOp. **Dynamic Cast** is typically used when iterating over the members of a group of diverse elements that must all be treated as if the group members are all of the same pType.

- to retrieve information about an object

  <Float> **Get Radius** <3D Object>

  <Float> **Magnitude** <Vector>

  <Vector> **Get Position** <3D Entity> <3D Entity>

  <CharacterBodyPart> **Get BodyPart** <Character> <String>

26.2 paramOps and Behaviors

paramOps are not activated in the same manner as BBs. paramOps do not have a bLink structure that supports activation. Instead, paramOps are only calculated when needed - when a BB or another paramOp requests the result of a paramOp.

If the value of a paramOp's pOut is requested:

1. the paramOp is triggered
2. the value of the pOut is calculated and updated
3. the value of the pOut is returned to the requesting pIn *and* pushed toward any other destinations that may exist for that pOut

paramOps also differ from BBs as follows:

- paramOps do not have any settings.
- paramOps may be created in any script attached to any type of object, the restrictions imposed on BBs that they may only be attached to objects of a certain CKClass do not apply.

However, paramOps *can* be directly synchronized with the activation flow by using **Op** or **Identity** (both in Logics/Calculator). See "Calculating a Value at a Specific Moment" on page 124.

26.3 Advanced paramOps

26.3.1 Order of pIns

The relevance of the order of inputs follows normal mathematical conventions; that is

<Float=50.0> **Multiply** <Float=10.0> <Int=5>

is equivalent to

<Float=50.0> **Multiply** <Int=5> <Float=10.0>

but

<Float=2.0> **Divide** <Float=10.0> <Float=5.0>

is *not* equivalent to

<Float=0.5> **Divide** <Float=5.0> <Float=10.0>

26.3.2 Calculating a Value at a Specific Moment

If you wish to calculate a paramOp at a specific moment in a script, you can use

- **Identity** to request the pOut and thus trigger the paramOp to perform the operation with the current pIns

- **Op** to perform the requested paramOp when activated by another BB (after selecting the Operation Type in the Settings for the BB)

Because **Identity** supports a variable number of pIns, **Identity** can be used to calculate several values at the same time.

26.3.3 pTypes <Angle>, <Float>, and <Percentage>

A paramOp will often involve a <Percentage> or <Angle> pType. These pTypes are actually considered to be of type <Float> by Virtools Dev but with unique representations:

- an <Angle> is represented as the number of turns and degrees
- a <Percentage> is represented as a value in the range 0% to 100%

Therefore, if a pIn of pType <Angle> has a value of 0:180 (0 turns, 180 degrees) then the value used for calculations is actually equal to 3.14159... (PI) because Virtools Dev must use *radians* rather than degrees when performing calculations.

NOTE There are 2*PI radians in 360 degrees.

Similarly, a <Percentage> of 50% has a value of 0.5 for calculations.

Therefore, wherever the you have a <Float> pIn or pOut, you can change the pType to <Angle> or <Percentage> for your convenience.

For example, if you create a paramOp

<Float> **Multiply** <Percentage> <Float>

with the values

<Float> **Multiply** <Percentage=50%> <Float=2.0>

the result will be 1.0 *not* 100%. For such an operation, it might be better to specify the pType of the pOut as <Percentage> for ease of comprehension.

27 ATTRIBUTES

Attributes are a means of adding information to elements. Attributes are a kind of parameter that belongs to an element instead of belonging to a script.

Virtools Dev includes numerous predefined attributes. You can also define your own attributes.

Only Behavioral Objects can have attributes. A Level is a BeObject and can have attributes. Level attributes are, effectively, global attributes.

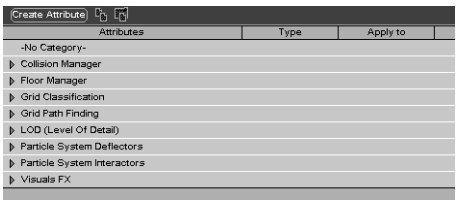
An attribute has a Name and may also have a Category. An attribute typically consists of one or more parameters that you can edit. However, some attributes (such as *ZBuffer Only*) can not be edited.

For example, an element with the Floor attribute is identified to the Behavioral Engine as an element that should be treated as a floor so that characters walk *on* the element and not *through* the element.

Attributes are added to elements during Author mode via the **Level Manager** context menu, via an element's **Setup**, or via the **Attributes Manager**.

The **Attributes Manager** (**27-1**) allows you to inspect and manage *which* elements have *what* attributes.

27-1 Attributes Manager



At run-time, Attributes can be:

- added via **Set Attribute**

- removed via **Remove Attribute**
- retrieved via **Has Attribute**, if the attribute exists on the element

All attribute BBs are in Logics/Attribute.

27.1 Attribute Shortcuts

Attributes are often used to store data values associated with an element. Typical uses for attributes include a player's score, a player's health, the characteristics of an object, etc.

If the value of an attribute is required by a script, then **Has Attribute** (Logics/Attributes) can be used to retrieve the value of the attribute. Alternatively, an attribute shortcut can also be used.

Attribute shortcuts are created by copying an attribute from the element's Attribute Setup (via the context menu) then performing a **Paste as Shortcut** operation (via the context menu) at the desired location in the **Schematic**.

Part 5 - Authoring in Virtools Dev

This is the part that everybody asks about - where can I find tutorials for Virtools Dev? This part contains two tutorials: a Quick Start for those of you just starting out in Virtools Dev, and a fun and informative tutorial on particles for those of you who have already some experience with Virtools Dev.

For the Quick Start, we would prefer you to peruse (and even read) the other parts beforehand. But if you are too impatient, we have taken care to make the Quick Start accessible to everyone and we sincerely hope we don't lose anyone on the way. Just make sure you read the other parts after!

The particles tutorial is just a sample of what is to come. Further tutorials will be posted to the Virtools website on a regular basis and you can even find some previews in the Virtools MiniSite.

The Virtools MiniSite is installed in the Documentation folder. However, if you chose not install this component, you will have to launch the installation program once more, and choose to install the MiniSite only.

Part 5 contains:

- **28 Quick Start** - a detailed walkthrough for creating your first composition
- **29 Particles** - an introduction to the world of Virtools Dev particle systems

28 QUICK START

28.1 Overview

The Quick Start tutorial is your introduction to the power and simplicity of Virtools Dev.

The Quick Start tutorial is structured very much like a real project in Virtools Dev. You will follow these steps:

1. organize resources
2. plan the content to be implemented - what do you want the composition (CMO) to do?
3. import the media; the models and characters that form and inhabit the world
4. arrange the scene
5. implement interactivity within the scene
6. test the scene
7. refine the scene based on the results of your tests
8. go back to (6) and continue to test and refine until you are satisfied that the scene meets the requirements you chose in (2)
9. release the composition

NOTE In actual production, you would plan the content before organizing the resources. However this is a Quick Start tutorial, so things are done a bit differently!

28.2 Organize Resources

This tutorial uses models contained in the data resource **VirtoolsResources**. **VirtoolsResources** are normally installed by default. However, if you chose not to install this component, you will have to launch the installation program

once more, and choose to install the Data Resources only.

To use the supplied models, **VirtoolsResources** must be open. **VirtoolsResources** are open if there is a tab labeled **VirtoolsResources** visible on the tab strip in the top-right region of the screen, next to **Building Blocks**. If the **VirtoolsResources** Data Resources are not open, perform the following steps:

1. from the **Resources** menu, select **Open Data Resource**
2. in the Virtools Dev program folder, open the Documentation folder, select **VirtoolsResources.rsc** and click **Open**.

   The **VirtoolsResources** tab will appear next to the **Building Blocks** tab, in the top-right region of the screen.

28.3 Plan the Content

Normally, planning the content is your responsibility. For the purposes of this tutorial, we have planned the content for you.

Using the models provided, create a world that can be explored by a player character controlled by the user. Provide multiple cameras (for different viewpoints) and support for collision detection between the character and the world.

28.4 Import Media

Once you have a clear plan, the next step is to import the scenery and actors into your world.

28.4.1 Importing the Scenery

Virtools Dev is an authoring platform, used to integrate media and add interactivity. The media can be of many forms: 3D object models, 3D character models with their animations, sounds, etc. In general, the media is created in some other application and then saved to or exported to a Virtools Dev readable format.

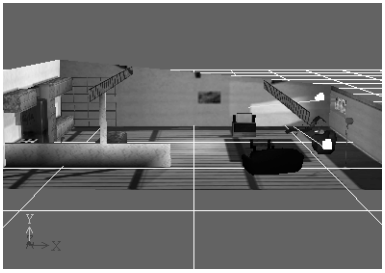
Normally, all your media are stored and organized in a data resource. A data resource is structured as a set of nested folders and files, just like the file system on the computer. When a data resource tab is selected, the window is divided in two sections. On the left hand side of the window is the directory structure for the data resource. On the right hand side of the window is a listing of the contents of the current folder.

Whenever you are instructed to select a file from VirtoolsResources, the location of the file is given in brackets. For example, in step 1 below you are instructed to select the file Apartment.nmo (3D Entities/Worlds). This means you should open the folder 3D Entities, then the sub-folder Worlds where you will find Apartment.nmo.

1. From **VirtoolsResources**, select Apartment.nmo (3D Entities/Worlds) and drag it into **3D Layout** to add the contents of the file to your scene.

   The file Apartment.nmo contains a scene created in a popular 3D modeling application and exported to Virtools Dev, as shown in **28-1** below. The scene contains a number of textured objects (such as walls, chairs, a TV, etc.) and three cameras. The scene does not use any lights; instead object materials were self illuminated. Self-illumination is a good technique to use to avoid CPU intensive real-time lighting.

28-1 Apartment.nmo, as seen in 3D Layout



28.4.2 Exploring the Scene in Author Mode

2. In **Level View** select KitchenTable (Global/3D Objects). Select **Orbit Target** to move around your scene while keeping Kitchen Table in the center of the view.
3. Use **Camera Dolly** to move the camera closer to and farther from the scene. Use **Camera Pan** to move up and down, and left and right.
4. When you are finished exploring, right-click in **3D Layout** and choose **Reset Current Camera Settings** to return to the view you had before you started exploring.

28.4.3 Adding a Character and Animations

5. From **VirtoolsResources** select Eva.nmo (Characters/SkinCharacters) and drag it into **3D Layout**.

   A female character, Eva, is added to the scene and the **Character Setup** opens. This character was created separately from the scenery.
6. Click the cross in the upper right corner of the **Character Setup** to close it, then use **Camera Dolly** to move forward so you can see the character better.

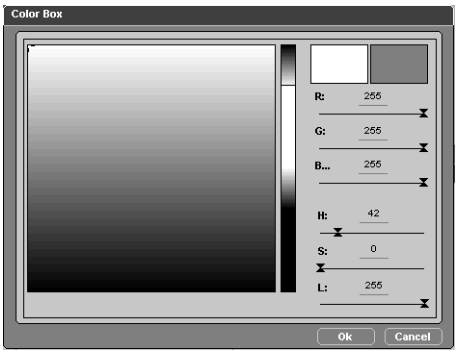
At the moment the character is not very visible, for there is no light in the scene. You can either add a light to the scene, or you can change the properties of the character to make it more visible.

You will modify the **Emissive** parameter value of the character's material to make the character more visible.

7. Right-click the character in **3D Layout** and choose **Material Setup (Eva_Material)**.

   Material Setup opens in the lower half of the screen.
8. In **Material Setup**, click the color box next to **Emissive**. In **Color Box**, enter values of 255 for **R**, **G**, **B** and **L** as shown in **28-2** and then click **OK** to close it.

28-2 Dialog box for choosing a color - Color Box



The character is now visible because you have adjusted the characteristics of the character's material so that it looks as if the material emits light. In other words, the character is self-illuminated.

9. Click the cross in the upper right corner of **Material Setup** to close the **Setup**.

10. In **VirtoolsResources**, hold the CTRL key down and click Run.nmo, Walk.nmo, Wait.nmo and WalkBackward.nmo (Characters/Animations/SkinCharacterAnimations/Eva) to select the four files. Drag these animation files onto the character. Release the mouse button only when you see a yellow bounding box (indicating that the character is selected) appear around the character.

    Each of these animations (Run.nmo, Walk.nmo, Wait.nmo and WalkBackward.nmo) were exported one at a time and separately from the character that you imported earlier in this tutorial.

28.5 Arrange the Scene

The next step is to arrange the scene, presenting the scene in the manner that you desire. Typically, this means creating and controlling lighting, creating a camera and setting the view point for the camera but can also include adding

or creating objects.

This scene does not need any lighting for all elements in the scene are self illuminated.

Three cameras were exported, with the scene, from the 3D modeling application (these cameras are used later in this tutorial). However, for the purposes of this tutorial, you will add another camera to the scene and set the view point for the camera.

28.5.1 Adding a Camera

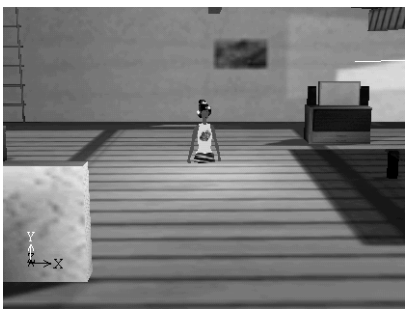
When authoring, Virtools Dev provides five default camera views (Perspective, Top, Front, Right, and Orthographic). However, the five default camera views will not be available when using a player such as the Virtools Web Player. Therefore you will *always* need to create a camera through which to view the scene if a camera is not already present.

11. Click **Create Camera**.

    A targeting camera is created and **Camera Setup** appears in the lower half of the screen.

12. In **3D Layout**, click **Camera Dolly** and move the mouse toward the top of the screen so that the camera, and therefore your viewpoint, approaches the character, as shown below in **28-3**.

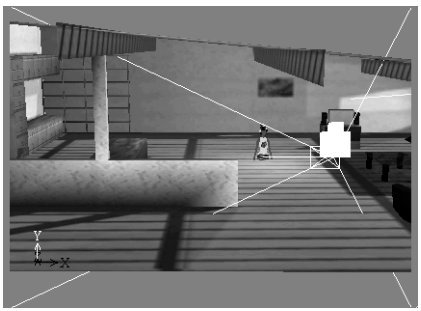
28-3 View after using Camera Dolly



13. Next, right-click in **3D Layout** and in the menu that appears choose **Select Camera** then **Perspective View**. Then select **Camera Dolly** and move your mouse slightly toward you until you see a white box, as in **28-4**.

    The white box you can see is the camera. The white lines represent the camera's view frustum. The yellow line links the camera to the camera target.

28-4 Camera you just created, represented by a white box



28.5.2 Activating the Camera at the Start of the Scene

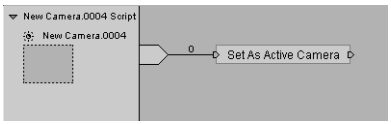
It is important to activate a particular camera at the start of a scene so that you can be sure that the user sees exactly what you want them to see.

14. From **Building Blocks**, select **Set As Active Camera** (Cameras/Montage) and drag it onto the camera you just created in the **3D Layout**. Release the mouse button only when you see a yellow bounding box surrounding the camera.

15. Click **Schematic**.

    You will see that a script has been created for the camera, as shown in **28-5**. By applying a building block (BB) to an element in **3D Layout**, you have just created your first script!

28-5 Script created by attaching Set As Active Camera to camera



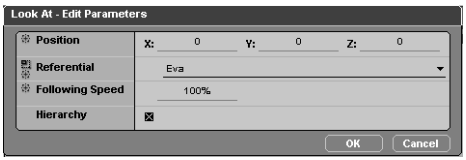
28.5.3 Targeting the Camera

At the moment the camera is fixed in position and orientation - if the character moves out of the camera's view frustum, you will no longer be able to follow the character. You will use a BB to make the camera always target the character, changing the orientation of the camera as necessary to keep the character centered in the camera's Field of View (FOV).

NOTE The camera you created is a targeting camera, meaning the camera always looks towards its target. However, the use of a target is no longer recommended and is only present for backward compatibility. But don't worry, you are about to learn a better way to make a camera always look at a certain object!

16. From **Building Blocks**, drag **Look At** (3D Transformations/Constraint) onto your camera in **3D Layout**, letting go when you see a yellow bounding box surrounding the camera. In the dialog box that appears, select Eva as **Referential** as shown in 28-6, and click **OK** to close.

28-6 Edit Parameters dialog box for Look At



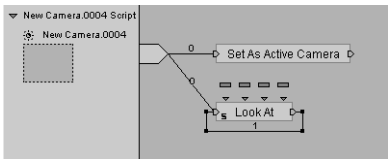
17. Look in **Schematic**.

    Look At has been added to the script you created earlier when you attached **Set As Active Camera** to your new camera. **Set As Active Camera** only needs to be activated once, whereas **Look At** needs to be activated every frame to always point at the character.

18. Click the Behavior Output (bOut), the pin on the right of **Look At**, and then and drag it to the Behavior Input (bIn), the pin on the left of **Look At**.

    A behavior loop is created, as shown in **28-7**.

28-7 **Look At** now linked and looped



19. Right-click the behavior loop and choose **Edit Link Delay**. (or double click on the link) Make sure the value for Link delay is 1, as shown in **28-8**.

28-8 Dialog box for Edit Link Delay

The camera you created will now look at the character every frame, adjusting its orientation if needed to always keep the character centered in the FOV.

28.6 Add Interactivity

So far you have set the scene and selected the camera to activate, but you are still missing the essential part - interactivity.

For the moment, interactivity in this tutorial will simply consist of making the character walk or run under the control of the keyboard.

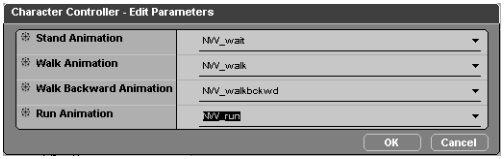
28.6.1 Controlling the Character

Although you have already attached four animations to the character, you have not yet implemented a mechanism for causing a given animation to be executed in response to user input.

The simplest way to control a character is using **Character Controller** (Characters/Movement). **Character Controller** executes animations in response to *messages*.

20. From **Building Blocks**, drag **Character Controller** (Characters/Movement) onto the character. Release the mouse button when you see a yellow bounding box surrounding the character.
21. In the **Edit Parameters** dialog box that appears, select the following parameters, as shown in **28-9**: NW_wait for **Stand Animation**, NW_walk for **Walk Animation**, NW_WalkBackward for **Backward Walk Animation** and NW_run for **Run Animation**.

28-9 Edit Parameters dialog box for Character Controller



28.6.2 Adding Keyboard Support

The character will now perform certain animations when receiving certain messages. You must now add a way of transmitting the required messages to the character in response to user input from the keyboard.

The following step illustrates one means for sending the desired messages in response to keyboard input.

22. From **Building Blocks**, drag **Keyboard Controller** (Controllers/Keyboard) onto the character, releasing the mouse button when you see a yellow bounding box surrounding the character.

    Keyboard Controller is automatically configured so you do not need to do anything further.

You can now control the character via the arrow keys on the numpad. **Key-**

board Controller translates the arrow keys that you press into messages. **Character Controller** receives each message and executes the animation corresponding to that message. Holding the INSERT key while moving the character enables the Run Animation.

28.7 Test

Now that you have a simple scene with some interactivity, the next step is to test it out.

NOTE Implement, test and refine, implement, test and refine, implement, test and refine! The "implement, test and refine" cycle allows you to rapidly develop new compositions a piece at a time. By checking your work often, you can never go too wrong!

NOTE Don't forget to regularly save your work. Virtools strongly recommends the following incremental save technique. When starting a new project, immediately name your composition and append a version number to the end of the file name (e.g. MyComposition00.cmo) then save the composition. Every time you achieve a short term goal in the development cycle, save the composition to a new version number (e.g. MyComposition01.cmo, MyComposition02.cmo, etc.). The multiple versions of the composition provide a complete development history and a ready source of backups in case you change your mind about a design decision or disaster strikes your working version of the composition.

28.7.1 Switching to Play Mode

23. Click **Play** in the bottom right corner of the Virtools Dev interface. Use the number pad arrow keys to move the character around. Remember to hold down the INSERT key while moving to make the character run.

28.7.2 Returning to Author Mode

24. Click **Pause** in the bottom right corner of the Virtools Dev interface.

NOTE Play and Pause are actually the same button, changing state as necessary.

28.8 Refine

The next step is to improve the scene. At the moment the character walks above the floor and is able to walk straight through supposedly solid objects, such as the desk, chairs, walls, etc.

28.8.1 Making the Character Stay on the Floor

There are several BBs that will cause a character to remain on the floor. In this tutorial you will use the simplest BB of this type: **Character Keep On Floor**.

For **Character Keep On Floor** to work, the behavioral engine must know that an element is to be treated as a floor.

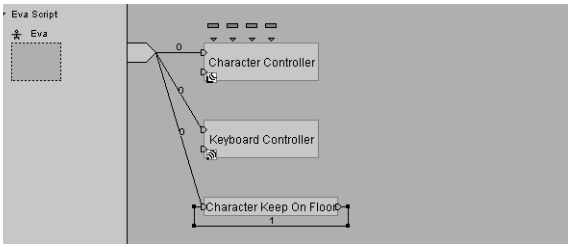
25. From **Building Blocks**, drag **Character Keep On Floor** (Characters/Constraint) onto the character, releasing the mouse button when you see a yellow bounding box surrounding the character.
26. Click **Schematic**, and you will find a script containing the three building blocks (**Character Controller**, **Keyboard Controller** and **Character Keep on Floor**) you dragged onto the character.

    Character Keep On Floor needs to be activated regularly so that the character always remains on the floor. In other words, it needs to be looped for it to work properly.

NOTE You have already created a loop for **Look At** in your camera script.

27. Click the Behavior Output (bOut), the pin on the right of **Character Keep On Floor**, and then drag it to the Behavior Input (bIn), the pin on the left of **Character Keep On Floor**.
28. A behavior loop is created, as shown in **28-10**. Right-click the behavior loop and choose **Edit Link Delay**. Make sure the value for Link delay is 1

28-10 Script to keep character on the floor



29. Next, right-click the floor in **3D Layout** and choose **3D Object Setup (Room_Sun)**. In the **3D Object Setup** that opens, click **Attribute** on the left side of the window to switch to attribute mode.

30. Click **Add Attribute**, and in the dialog box select *Floor* (Floor Manager), as shown in **28-11**.

28-11 Dialog box for Add Attribute

31. Click **Add Selected**, then **Close**.

    You can see that the attribute *Floor* has been added to this object, as it now appears in the attribute list, as shown in **28-12**. Finally, close **3D Entity Setup**.

28-12 Floor Attribute has been added successfully

Name	Category	Value	
Floor	Floor Manager	Faces;TRUE;0;FALSE;TRUE	

32. Click **Reset IC**, then **Play**.

    The character now recognizes the floor.

28.8.2 Adding Simple Collision Management

There are also various ways to implement collision detection in Virtools Dev. In this tutorial, you will use **Object Slider** (Collisions/3D Entity). **Object Slider** makes the character slide on any objects it comes into contact with - as compared to abruptly stopping the character on contact.

33. From **Building Blocks**, drag **Object Slider** (Collisions/3D Entity) onto the character, letting go when you see a yellow bounding box surrounding it. Accept the default parameters and click **OK** to close the dialog box.
34. Click **Reset IC**, then **Play**.

    The character still walks through objects. Why?

28.8.3 Declaring Objects as Obstacles

Although the character now has a BB preventing it from colliding with objects, the character still walks through chairs, the table and other objects - even walls.

Again, just as with floor management, **Object Slider** requires that you identify the elements of your composition that are to be treated as obstacles. However, for **Object Slider** you do not use attributes but instead place your obstacles into a Group.

35. In **Level View**, click 3D Objects (Global) to open the folder, then right-click 3D Objects and choose **Select Children**.

    You have just selected all objects in the scene. If you look at **3D Layout**, bounding boxes are around all objects in the scene.

36. Right-click the selection, and choose **Place Selected in New Group**.

    A new group is created that references all 3D Objects in the scene.

37. Rename this group from New Group to Obstacles.

NOTE You can rename any element in the Level Manager by right-clicking the element and choosing Rename. All references to the renamed element, throughout your composition, are automatically updated to reflect the new name.

38. In **Schematic**, right-click **Object Slider** and choose **Edit Parameters**. In the dialog box that appears, select Obstacles in the **Group** pull-down list, as shown in **28-13**.

    Object Slider will now prevent the character from walking through the objects you referenced in the group Obstacles.

28-13 Edit Parameters dialog box for Object Slider

28.9 Test Again

Before you go on, you should test whether the collision detection system you have implemented works as you intended.

28.9.1 Switching to Play Mode

39. Click **Reset IC** and then **Play**. Use the number pad arrow keys to move the character around, holding down the INSERT key to make the character run. Try to make the character walk through the objects - and walls - of the scene.

28.9.2 Returning to Author Mode

40. Click **Pause** to return to Author mode.

28.10 Refine Again

The character no longer walks through supposedly solid objects, but there are still a number of improvements that can be made. For example, three cameras were exported with the scene but none are currently used. You will now create a script that enables you to switch between cameras during play.

28.10.1 Dynamically Switching Cameras

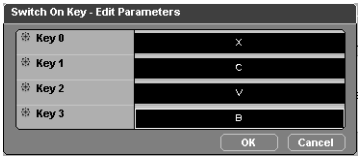
The next step is to add keyboard control over the cameras in the scene.

41. In **Schematic**, ensure that you can see the script New Camera.0004 Script. From **Building Blocks** drag the following BBs into the script (as shown in **28-14**): **Switch On Key** (Controllers/Keyboard), **Parameter Selector** (Logics/Streaming) and **Set As Active Camera** (Cameras/Montage).

28-14 Script for Switching cameras - work in progress

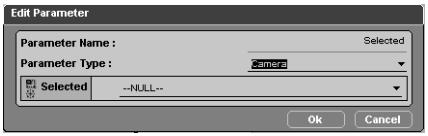
42. Right-click **Switch On Key** and choose **Construct -> Add Behavior Output**. Repeat this operation so that you have a total of four bOuts (one bOut for each camera). Right-click **Switch on Key** and choose **Edit Parameters**. Configure the pIns as follows (shown in **28-15**): X for **Key 0**, C for **Key 1**, V for **Key 2**, B for **Key 4**.

28-15 Edit Parameters dialog box for Switch on Key



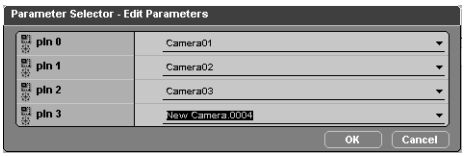
43. Right-click **Parameter Selector** and choose **Contruct -> Add Behavior Input**. Repeat this operation so that you have a total of four bIns. Right-click the parameter output (the pOut, just under the V in the lower left hand corner) of **Parameter Selector**, and choose **Edit Parameter**. In the **Edit Parameter** dialog box, change **Parameter Type** from Float to Camera (as shown below in **28-16**), then click **OK**.

28-16 Edit Parameter dialog box, used to change parameter types (pType) for variable BB Parameter Selector



44. Right-click **Parameter Selector** and choose **Edit Parameters**. Configure as follows (shown in **28-17**): Camera01 for **pIn 0**, Camera02 for **pIn 1**, Camera03 for **pIn 2**, New Camera.0004 for **pIn 3**.

28-17 Edit Parameters dialog box for Parameter Selector



45. Right-click the **Set As Active Camera** you added a moment ago and choose **Add Target Parameter**.

Now all that remains is to add behavior links between these building blocks (BBs) and a parameter link from a pOut on one BB to a pIn on another BB (see **28-18**).

46. Click the bOut and bIn you want to link.Working from the left, link the bOut of **Set As Active Camera** to the top bIn of **Switch On Key**. Next

link the bOuts of **Switch On Key** to bIns of **Parameter Selector**. Link the bOut of **Parameter Selector** to bIn of **Set As Active Camera**.

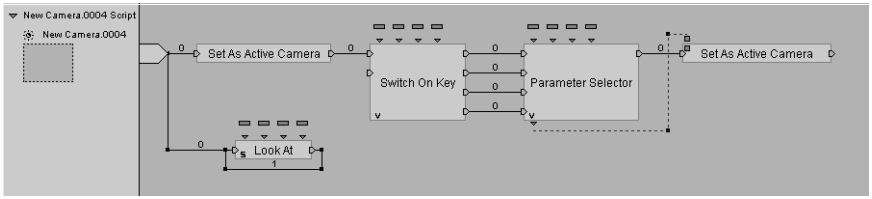
You have completed all behavior links (bLinks). The last thing that remains is a parameter link, to pass information from one BB to another.

NOTE You can also use the keyboard shortcut L to activate the **Link** button.

47. Link the pOut of **Parameter Selector** to the target parameter of **Set As Active Camera**.

    Your script should look like **28-18**.

28-18 Script for selecting cameras



NOTE **Switch on Key** only needs to be activated once; Switch on Key remains active until it is deactivated. For more information on the different types of BB, See "BB Processing" on page 113

28.11 Test Again

Once again, you must test whether your implementation of camera switching works as expected. Don't forget to save your composition too, preferably using the technique presented in "Test" on page 141.

28.11.1 Switching to Play Mode

48. Click **Reset IC**, then **Play**. The scene should start off using the camera you created yourself (New Camera.0004). Use the Z, X, C and V keys to switch between cameras.

28.11.2 Returning to Author Mode

49. Click **Pause**.

28.12 One Last Refinement

There is one last refinement you could make - you could make all cameras look at the character. This last refinement you can try to do by yourself. Here is a hint: the simplest way is to attach **Look At** to each camera. Perhaps there are other ways - it is up to you to find out!

But don't worry if you can't figure it out right away - you will find a completed version of this tutorial QSComplete.cmo in the Virtools Dev Program folder: Documentation/Cmos/FinishedCmos.

28.13 Export Content

28.13.1 Saving Your Hard Work

You can save your work in two different formats: CMO or VMO.

CMO is an editable version of your work that you can open in Virtools Dev and view in players such as the Virtools Web Player.

VMO is a view only file format - VMO files are for players only and cannot be opened in Virtools Dev.

You should always save you work (*a final time* - you have been saving intermediate versions, haven't you?) in a CMO first.

50. From the **File** menu, select **Save Composition**. Choose a name and location for your composition.

    You will be able to re-open this composition in Virtools Dev and edit it.

Once you have saved your work as a CMO, you may wish to save it as a VMO.

51. From the **File** menu, select **Export to Virtools Player**. Choose a name and location for your composition.

    This file can now only be opened by a Virtools player, such as the Virtools Web Player.

28.13.2 Sharing Your Content With Others

When you plan to share your composition with others, for example on a web page, there are two advantages to using the Export to Virtools Player function.

The first advantage is that no one can then look at the details of how you implemented the interactivity within your composition, or extract your media from the composition for (potentially) unauthorized re-use.

The second advantage is that the file size is smaller - always a positive factor for Internet delivery. To view your work:

52. use the Web Player - drag the.cmo file into an Internet Explorer or Netscape Communicator window.
53. use the Create Web Page command from the file menu to create a web page, complete with the html tags needed to start the Virtools Web Player.

28.14 If You Encountered Any Difficulties...

You will find two completed versions of this exercise, in the folder: Documentation/Cmos/FinishedCmos:

1. QSComplete.cmo
   for use in Virtools Dev

   and
2. QSComplete.vmo
   to view with the Virtools Web Player

28.15 Congratulations

You have successfully completed the Quick Start tutorial! In this tutorial, you have learned how to do a number of things:

- add media, in the form of models, a character and animations to a scene
- create a camera, target the camera, and activate the camera at the beginning of a scene
- attach BBs to objects, and add BBs to existing scripts
- edit a BB - both the parameter type (pType) and the parameters themselves
- link BBs to other BBs (activation flow) via behavior links
- link parameter outputs (pOuts) of one BB to the parameter inputs (pIns) of another BB (data flow) via parameter links
- open and close Entity Setups
- add attributes to an element (just a reminder - attributes are parameters that belong to an element rather than to a script)
- create a group and add elements to that group
- save your file in an editable format
- export your file to a Player only format

29 PARTICLES

29.1 Introduction

In this tutorial you will learn how to generate sophisticated visual effects like sparks, smoke, snow, fireworks, and clouds of dust using the Virtools Dev particle systems.

The Virtools Dev particles system can be divided into four distinct parts:

1. emitters
2. particles
3. deflectors
4. interactors

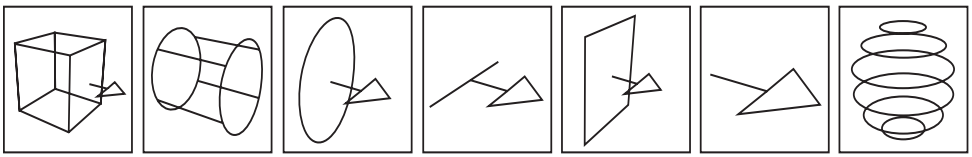
29.2 Emitters

An emitter is a source of particles. Many different emitters are supported (spherical, disc, point etc) - choose the type of emitter according to the desired effect. Most emitters can be applied to any 3D Entity (and, therefore, to their descendent classes also) but are typically applied to 3D frames. However, the **CurveParticleSystem** emitter can only be applied to Curves and the **ObjectParticleSystem** can only be applied to 3D Objects.

Each type of emitter is identified by a unique icon.

NOTE The particle system BB determines the type of emitter

29-1 Emitters, from left to right - Cubic, Cylinder, Disc, Linear, Planar, Point and Spherical



29.3 Particles

Particle systems have many different parameters that control how particles behave. For example, you can control a particle's lifespan, speed, weight, and color. The particles themselves can be of different styles: simple points, lines, textured sprites (animated or still), or 3D objects.

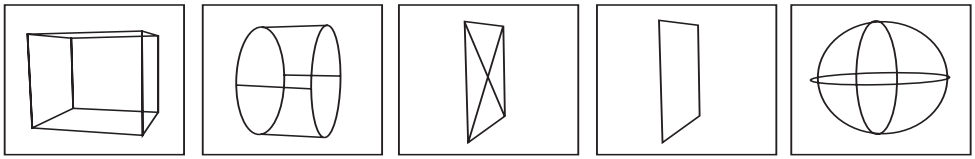
29-2 Particles, from left to right - Point, Line, Sprite, Orientable Sprite, Radial Sprite and Fast Sprite

29.4 Deflectors

Deflectors are types of obstacles that modify the path of particles by getting in their way and 'bouncing' them in another direction. There are different types of deflectors (spherical, cylinder, infinity, plane, etc.) that can be applied to frames or any 3D object.

Each type of deflector is identified by a unique icon.

29-3 Deflectors, from left to right - Box, Cylinder, Infinite Plane, Plane and Sphere



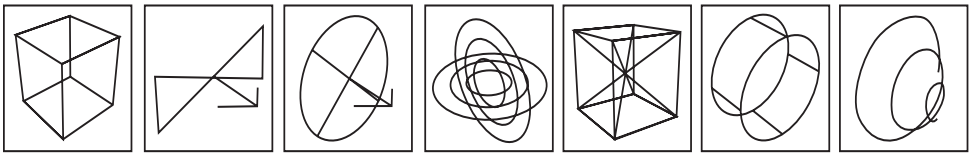
29.5 Interactors

Finally, there are interactors (wind, gravity, magnetic fields, and mutation boxes) that modify the behavior, course or appearance of particles. With the

exception of gravity or atmosphere (which can be applied to all types of objects, including emitters), interactors are applied to 3D frames only.

Each type of interactor is identified by a unique icon.

29-4 Interactors, from left to right - Disruption, Global Wind, Local Wind, Magnet, Mutation, Tunnel and Vortex



29.6 Configuring a Particle System

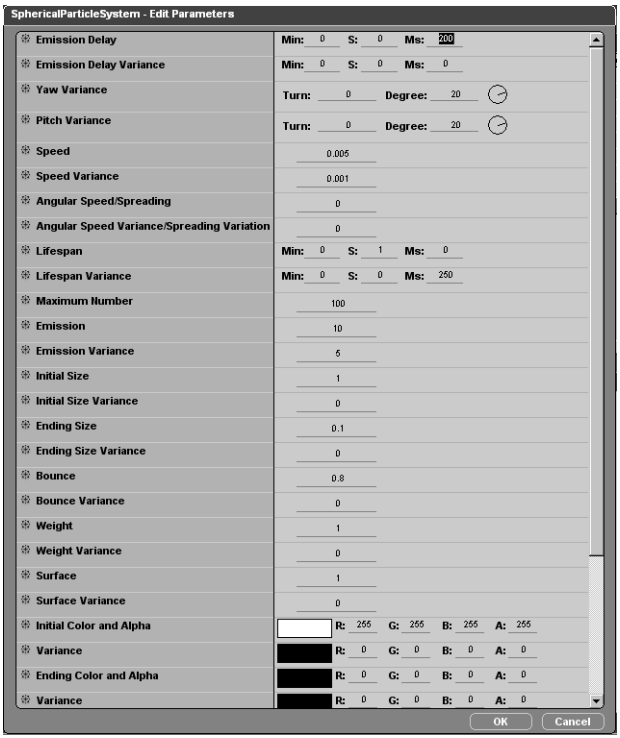
Particle systems can be interactively controlled. For example, you can activate a particle emitter, alter the orientation of a particle emitter, activate a deflector or interactor, or even modify the characteristics of the particles while the composition is playing.

A particle system is configured via the **Edit Parameters** dialog box. The **Edit Parameters** dialog box, shown in **29-5**, is displayed when:

- creating a particle system by dragging a particle system Building Block (BB) onto a 3D object or 3D frame in **3D Layout**
- creating a particle system by dragging a particle system BB onto a 3D object or 3D frame in **Level Manager**
- right-clicking a particle system BB in **Schematic**

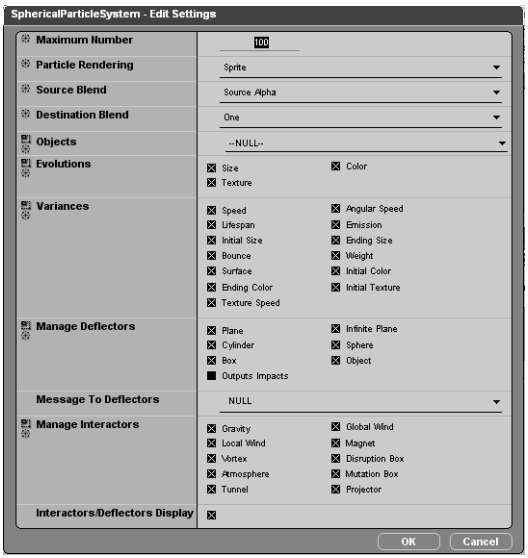
Typically, you will only use a small number of the available parameters. The unused parameters can be hidden via the **Edit Settings** dialog box (opened by right-clicking the BB in **Schematic**), shown in **29-6**. Hiding the unused parameters via the **Edit Settings** dialog box removes the hidden parameters from display *and* from further processing - thus reducing the CPU load at runtime and helping to maintain a satisfactory frame rate.

29-5 Edit Parameters dialog box for a particle system



NOTE Particles use a lot of system resources. To maintain a satisfactory frame rate, keep the number of emitters to a minimum and avoid creating large numbers of particles from each emitter. The default number of particles is 100. In some cases you may need to use more, but generally you can obtain a satisfactory effect by fine-tuning a few of the parameters.

29-6 Edit Settings dialog box for a particle system



29.7 StartCmos and FinishedCmos

At the start of exercises 1, 2, 3, and 5 you will be instructed to open a file. You will find these files in the Virtools Dev program folder, in the following location: \Documentation\Cmos\StartCmos.

You can find the completed version for all exercises in \Documentation\Cmos\FinishedCmos.

29.8 Exercise 1 - Particle System Basics

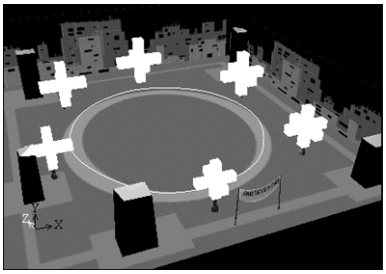
In this exercise you will learn the basics of creating particle systems.

29.8.1 Start

1. Open the file ParticlesExercise01.cmo.

   A small town, in the middle of the night, is displayed as shown in **29-7**. Use the camera navigation tools if you wish to change the viewpoint. The 3D frames necessary for the emitters are already present, as is a curve used in a later exercise.

29-7 3D Frames ready for particle system BBs

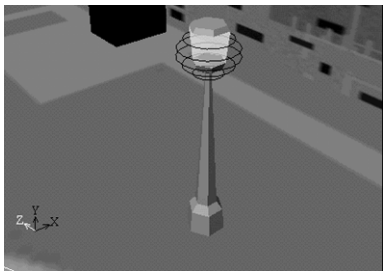


29.8.2 Placing an Emitter

2. In **Level Manager**, select LampFrame_01 (Global/3D Frames).
3. From **Building Blocks**, drag **SphericalParticleSystem** (Particles) onto LampFrame_01 in **Level Manager**.
4. The **Edit Parameters** dialog box for this BB opens - just click **OK** for now.
5. Click **Zoom on Selection**.

   You can see that the 3D frame now has an orange wire-frame spiral representing the emitter, as shown in **29-8**.

29-8 Particle system emitter represented by an orange colored spiral



NOTE Sometimes it can be easier to drag a BB onto an object in **Level Manager** than onto an object in **3D Layout**. In this example, the 3D frame is on the inside of the street light. If you attempted to drag the BB into **3D Layout** and onto the 3D frame you would be unsuccessful. Instead of attaching the particle BB to the 3D frame, the BB would try to attach to the street light - and would have generated an incompatibility error message.

29.8.3 First Few Particles

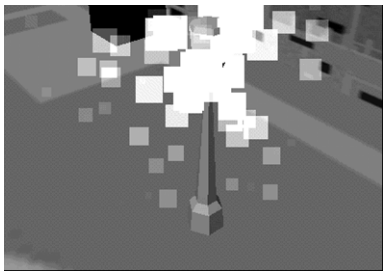
6. In the lower right part of the screen, click **Reset IC**, then **Play**.

   You can see particles being emitted.

7. Click **Zoom on Selection**, then **Camera Dolly** - slowly moving your mouse back towards you to bring more of the particle effects into view.

   The particles are in the form of small white squares, as shown in **29-9**. By default, these sprites do not have a texture.

29-9 Particles represented by small white squares - by default, particles are not textured



29.8.4 Changing a Basic Parameter: Adding Color

8. In **Schematic**, right-click **SphericalParticleSystem** and choose **Edit Parameters**.

NOTE It is not necessary to click **Pause** each time you want to change a parameter, opening the **Edit Parameters** dialog box pauses the composition. On closing the dialog box, play automatically continues. You only need to click **Reset IC** if you add a new particle system or other BB. You do not have to click **Reset IC** if you just change some parameters.

9. In the **Edit Parameters** dialog box, under the parameter **InitialColorAndAlpha**, click the white box. In the **Color** dialog box that opens, choose a bright color. Click **OK** to close.

   The particles are no longer white, they are now the color you just chose.

10. Re-open the **Edit Parameters** dialog box and choose a different color for **EndingColorAndAlpha**.

    The particles change from the first color to the second as they move away from the street light.

11. To allow each of these colors to vary, open the **Edit Parameters** dialog box once again and adjust the **Variance** for each. To see these changes more easily, alter the particle **Lifespan** from 1000 ms to 2000 ms.

NOTE Alpha is the name given to transparency, encoded in 8 bit grayscale for 32 bit color and as a single bit mask for 16 bit color. When Alpha is set to black, particles are entirely transparent, and when Alpha is set to white, particles are completely opaque. Use Alpha to create translucent systems, such as smoke.

29.8.5 Texturing Particles

12. From **VirtoolsResources**, drag the texture Spark.jgp (Textures/Particles) into **3D Layout**.
13. In **Schematic**, open the **Edit Parameters** dialog box for **SphericalParticleSystem**. Choose Spark in the **Texture** parameter pull-down menu.
14. Click **OK**, then **Play**.

The particles now have a texture and appear as small flakes. These textures are static, but you will see later that you can also use animated textures.

NOTE If you have graphics editing or viewing programs installed and associated with the texture file types, you can edit or view the texture by double-clicking the texture in the data resource. Virtools Dev automatically opens the file in the associated graphics program.

29.8.6 Configuring Speed, Lifespan, and Size

15. In **Level Manager**, repeat the steps above to create identical particle systems on the 3D frames on the other street lights.

Now you will change parameters, one at a time, so that you can understand the effects you can achieve with these simple changes.

16. Adjust the parameter **Speed** for each 3D frame, then click **Play**.

    With a **Speed** of 0.05 you will see very fast particles, like cannon balls. With a **Speed** of 0.0001 you will see something that resembles a halo of multi-colored magma.

17. Now adjust the parameter **Lifespan**. Choose values between 10 and 2500 ms.

    With **Lifespan** you can obtain results such as a drone of particles surrounding the emitter, clouds of explosions, swarms of gnats, etc.

18. Finally, adjust the sizes of each particle system, either with **Initial Size**, **Ending Size** or both.

    With **Initial Size** and **Ending Size** you can obtain astonishing results: from flows of lava to vibrating halos, as illustrated in the completed version of this exercise Particles01Finished.cmo.

NOTE Particles with a high **Speed** value do not require a long **Lifespan**. High speed particles are quickly out of camera range. For optimum performance, a particle's **Lifespan** should only be long enough for the particle to exit the camera's view frustum.

29.8.7 Variance and Other Parameters

The principles shown above also apply to the other particle system parameters, such as the parameters that specify the initial size and ending size of particles, the total number of particles, etc.

All of these parameters also have a **Variance** (as you may have noted when changing the particle colors). The **Variance** defines the range over which a parameter's value can vary.

For example, if you set a **Lifespan** of 500 ms with a Variance of 100, the lifespan of particles can vary between 400 ms and 600 ms. Some of the possible effects are shown in **29-10**.

29-10 Particle effects achieved by using Variance

29.8.8 Conclusion

In this exercise you learned how to modify the basic parameters of a particle system: **Color**, **Texture**, **Speed**, **Size** and **Lifespan**.

29.9 Exercise 2 - Moving an Emitter and Adding Interactors

Now that you know the basics of particle systems, you are ready to create moving particle systems and particle systems that are dynamically modified at run-time.

29.9.1 Creating a Moving Emitter

Emitters, deflectors and interactors can be stationary or moving. First, you will create a moving emitter.

1. Open the file Particles02.cmo.

   All the street lights are now working and there is a pink car (with two 3D frames attached to it) that is used in this exercise.

2. In the **Level Manager**, select PinkCar_Frame (Global/3D Frames).

   This 3D frame is at the back of the car.

3. From **Building Blocks**, drag **PointParticleSystem** (Particles) onto PinkCar_Frame in **Level Manager**.

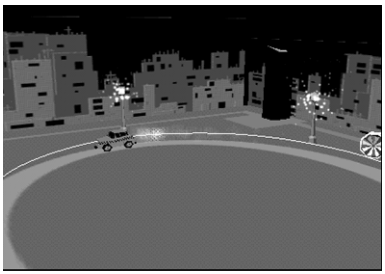
4. In the **Edit Parameters** dialog box that appears, select Star as the **Texture**.

5. To obtain a better visual effect, change **Lifespan** to 800 ms, **Speed** to 0.01 ms, and **Initial Size** to 2 with a **Variance** of 1. You can also change the **Variance** for **InitialColorAndAlpha** and **EndingColorAndAlpha**.

6. Click **OK**, **Reset IC**, then **Play**.

   The pink car now leaves a trail of (multi-colored) stars after it, as shown in **29-11**.

29-11 Moving emitter



29.9.2 Adjusting Gravity

Gravity is an attribute that modifies the behavior of all particles in the scene. *Gravity* is global, therefore you can only have one gravity parameter for the entire scene.

7. In **Level Manager**, double-click Floor (Global/3D Objects).
8. In the **3D Object Setup** that appears, click **Attribute** in the left hand column, then **Create Attribute**.
9. In the **Create Attribute** dialog box that appears, select *Particle Gravity* (Particle Systems Interactors), click **Add Selected**, then **Close**.

   You will see that the 3D object Floor now has the attribute *Particle Gravity* with a value of -0.0001.
10. Click **Reset IC**, then **Play**.

    All particles in the scene are now pulled down by the gravity affect. This is especially true for those particles with a slow **Speed**.
11. Double-click the parameter value -0.0001. In the **Edit Parameters** dialog box that appears, change the value to -0.0002.
12. Click **OK**, **Reset IC**, then **Play**.

    Though the change in *Gravity* seems small, the effect is great on all of the street lights. Only particles whose **Speed** is relatively high seem to be spared the effects of *Gravity*.

It is best to modify particle system parameters and attributes gradually. For example, if you set *Particle Gravity* to -5, you will not see any particles - the force of *Gravity* is so powerful that it pulls the particles out of view as soon as they are created. Conversely, you can enter positive values for *Gravity* - the particles will rise instead of falling.

You can also change the **Weight** for individual particle systems via the **Edit Parameters** dialog box. Changing the **Weight** allows you to differentiate between particle systems if you have more than one emitter in a scene. For example, if you enter a *negative* value for **Weight** when the value of *Particle*

Gravity is *positive* then your particles will sink rather than rise. Using **Weight Variance** values that cause **Weight** to have both positive and negative values allows you to have both falling and rising particles from the same particle system.

NOTE *Particle Gravity* and *Particle Atmosphere* are the only attributes of the Interactor type that can be applied to any entity or object in Virtools Dev - including particle emitters. All other attributes must be applied to 3D frames.

29.9.3 Adding Wind

You may have noticed that the fan found on the right side of town doesn't seem to have an effect, even though it is turning. For the fan to affect the particles, you need to add a *Particle Global Wind* or *Particle Local Wind* attribute.

13. In **Level Manager**, double-click Fan_Frame (Global/3D Frames).

    This 3D frame has been hierarchically attached to the fan.

14. In the **3D Frame Setup** that opens, click **Attribute**, **Create Attribute**. In the **Create Attribute** dialog box, select *Particle Local Wind: Force/Decay* (Particle Systems Interactors), click **Add Selected** then **Close**. Click **Play**.

    Nothing happens. This is because the range of the interactor is not great enough to affect any particle systems.

15. Make sure you have Fan_Frame selected, then click **Zoom on Selection**.

    The green disc with an arrow shows the presence of a dynamic attribute (in this case *Particle Local Wind: Force/Decay*). Although its range is not great enough, you can easily change this by using the **3D Layout** manipulation tools (as you would for any object).

16. Use **Camera Dolly** to move back, then click **Select and Scale** to increase the deflector's range so that it can interact with the passing pink car and the two closest street lights. Finally **Play** the scene to verify that the desired interaction exists.

NOTE In addition to having a **Weight** parameter that influences how particles are affected by *Particle Gravity*, you can also use the **Surface** parameter to adjust

how *Particle Wind* (*Local* or *Global*) affects individual particles. The **Surface** parameter adjusts the apparent surface area of a particle – the larger the surface area, the greater the effect of the Particle Wind parameter. A value for the **Surface** parameter that is of the opposite sign to the *Particle Wind* attribute creates an inverse reaction and the particle moves in the opposite direction of the *Particle Wind*.

29.9.4 Multiple Attributes on a Single 3D Frame

Virtools Dev allows you to add several attributes to a single 3D frame. A 3D frame could have an emitter, deflector and an interactor attached to it. It is also possible to have several attributes of the same type (all deflectors or interactors) such as *Local Wind* + *Magnet* + *Vortex* + *Gravity* attached to a single 3D frame.

NOTE Virtools strongly recommends against combining multiple particle attributes on a single frame. The particle system settings can conflict with each other, can use significant processing resources, and are almost impossible to calibrate as desired. However, you *can* change the type of attribute (deflector or interactor) during run-time by using the BBs contained in the Logics/Attributes folder.

17. In **Level Manager**, select Fan_Frame (Global/3D Frames). You will now add an emitter to this 3D frame, but first you should deactivate the attribute *Particle Local Wind*.
18. In the **Attribute** part of **3D Frame Setup**, double-click the **Value** cell of *Particle Local Wind: Force/Decay* and in the **Edit Parameter** dialog box ensure that the **X** and **Y** values are set to 0.
19. From **Building Blocks**, add **DiscParticleSystem** (Particles) to Fan_Frame in the **Level Manager**.
20. In the **Edit Parameters** dialog box that appears, set the following parameters: **Speed** 0.1, **Lifespan** 800, **Weight** 0.5 with a **Variance** of 0 for each parameter. To generate a jet of particles, set **Yaw Variance** and **Pitch Variance** to 0. Finally, choose a **Texture**, such as Spark. Click **Play** to test your composition and refine your parameters in real-time if you think they need to be changed.

21. In the **3D Frame Setup**, re-activate the attribute *Particle Local Wind* for Fan_Frame by double-clicking the **Value** cell, and setting a value of 0.001 for **X** in the **Edit Parameter** dialog box. Click **Play**.

    You can now see the influence the fan has on the particles.

22. Press F2.

    The view is now the perspective of a camera mounted at the front of the pink car. Press F2 to switch between cameras.

29.9.5 Magnet and Other Attributes

To finish this exercise, you will add a particle magnet to the center of the scene.

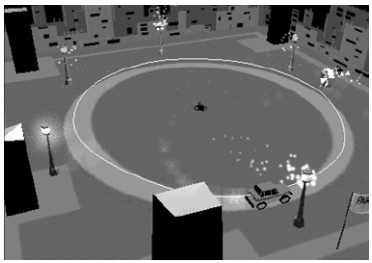
23. In **Level Manager**, double-click CenterFrame (Global/3D Frames) to open its setup. Click **Attribute**, **Create Attribute**, and in the **Create Attribute** dialog box select *Particle Magnet* (Particle System Interactors), click **Add Selected**, **Close**, then **Play**.

    The change is not very obvious due to all the other interactors present. To counter them, you will have to adjust the parameter value for *Particle Magnet*.

24. Increase the value of *Particle Magnet* from 1 to 5. Particles are now drawn to the middle of the scene, despite the presence of other interactors.

The other attributes in Particle System Interactors function in much the same way. *Particle Vortex*, *Particle Mutation Box*, etc. are applied to 3D frames and adjusted using the same process. Now you just need to try them out to obtain effects similar to those shown in **29-12**!

29-12 Same particle system, but different effects obtained by editing the parameters



29.9.6 Conclusion

In this exercise you learned to use more advanced emitter parameters such as **Weight**, and **Surface** in conjunction with dynamic attributes - that is interactors. You also learned to place these attributes, such as *Particle Local Wind: Force/Decay*, *Particle Gravity* on moving and stationary 3D frames.

29.10 Exercise 3 - Deflectors

Deflectors are another part of the Virtools Dev particle system. Deflectors are obstacles or barriers that stop particles or modify their path. As with interactors, you create deflectors either via the **3D Frame Setup** or by using the context menu in **Level Manager**. The following parameters apply to all deflectors:

- **Response**: the factor by which a particle will be slowed down or sped up by contact with an obstacle.
- **Friction**: the factor by which a particle's speed is affected when the particle bounces off of an obstacle.
- **Density**: similar to a filter, the percentage of particles that pass through the obstacle rather than bounce off of the obstacle. At a **Density** of 0% all particles pass, at 50% only one in two particles pass, at 100% all particles bounce. The exact equation for Density can be found on the particle systems' help page.

For example, it is possible to adjust these attributes to create a deflector that lets through just 20% of a particle system while violently rebounding the rest.

29.10.1 Start

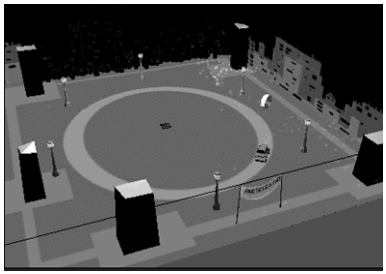
1. Open the file Particles03.cmo.

   This is the same scene as before but with a few changes: there are now particle systems at the front of the pink car and on the rotating fan. The street lights no longer have particle systems - this is to lighten the CPU load. Finally, the *Particle Gravity* interactor has been applied to the Level.

2. Click **Play**.

   Particles pass through buildings and through the ground as there is a slight gravitational pull.

29-13 Creating particle system deflectors



29.10.2 Creating a Deflector

3. In **Level Manager**, double-click FloorFrame (Global/3D Frame) to open its **Setup**. Click **Attribute**, **Create Attribute**, and in the **Create Attribute** dialog box, select *Particle Infinite Plane Deflector* (Particle Systems Deflectors), click **Add Selected**, **Close**, and then **Play**.

   You will see that the particles now rebound off the ground with force. You will now adjust the parameters for the *Particle Infinite Plane Deflector* attribute.

4. Double-click the cell under **Value** for *Particle Infinite Plane Deflector* and change **Response** to 0.7 and **Friction** to 0.4.

   You can also change the value of the parameter **Bounce** for each emitter (by right-clicking the appropriate BB). The parameter **Bounce** allows you to have unique qualities for each particle system (as you have already seen with **Weight** and **Surface** in the prior exercises).

29.10.3 Placing and Resizing a Deflector

At present, particles are still leaving the scene on all four sides. To stop the particles from leaving the front and the left of the scene, you must place deflectors on the two frames on these sides of the terrain.

5. In **Level Manager**, select BoundaryFrame01 and BoundaryFrame02 (Global/Frame) by pressing CTRL and clicking them. Right-click the selection and choose **Add Attributes**. In the **Create Attribute** dialog box, select *Particle Plane Deflector* (Particles Systems Deflectors), click **Add Selected** then **Close**.

   Each 3D frame has changed shape and is now colored yellow, indicating that the 3D frame now has the attribute *Particle Plane Deflector*.

NOTE Creating a multi-selection in **Level Manager** and choosing **Add Attributes** when you right-click the selection is a handy way to give the same attribute to several objects at the same time.

6. Select BoundaryFrame01 only, click **Zoom on Selection**, then **Camera Dolly** - dragging the mouse slowly towards you to move the camera back.

   Note that the deflector is not oriented as required - the deflector is parallel to the ground, whereas it needs to be at right angles to act as a barrier to the particles. You will have to orient the deflector the right way.

7. Click **Select and Rotate** and ensure that the **X** axis (red and facing right) constraint is selected. Rotate the deflector approximately 90 degrees.

   Now you must enlarge the deflector so that it covers the whole side.

8. Click **Select and Scale**, ensuring you are in **Uniform Scaling** mode with the **X** axis constraint still selected. Move your mouse so the deflector is approximately the same length as one side of the terrain.
9. Do the same for **BoundaryFrame02**, but along the **Z** axis this time. **Play** the scene.

   You will see that the particles now bounce back off the deflectors, towards the center of the terrain.

29.10.4 Placing a Deflector on an Object

The buildings in the background of the scene, called City_Background01 and City_Background02 are 3D objects composed of a dozen faces each. The low number of faces makes them ideal for applying the **Object Deflector** attribute.

10. In **Level Manager**, select both City_Background01 and City_Background02 (Global/3D Object), right-click the selection and choose **Add Attributes**. In the **Create Attribute** dialog box, select *Particle Object Deflector* (Particle Systems Deflectors), click **Add Selected** then **Close**.

NOTE *Particle Object Deflector* is the only deflector you can use with meshed objects.

The terrain is now completely enclosed for particles.

11. To finish, select all five buildings and the car, apply the attribute *Particle Object Deflector* to them also, then click **Play**.

29.10.5 Editing Particle System Attributes

It can sometimes be difficult to remember *which* object has *what* attribute. You can use the **Attributes Manager** to keep track of attributes and to edit them.

12. From the **Editors** menu, select **Attributes Manager**, which opens in the lower half of the screen.

On the left you will find all attribute types. As you select each attribute on the left, the elements in your current scene with that attribute are listed on the right.

13. Open Particle Systems Deflectors and select *Particle Object Deflectors*.

    You will see all the objects in the scene that have this attribute. You can easily edit, copy or paste any parameters using the context menu, or you can use the appropriate icons in the **Attributes Manager** toolbar.

29.10.6 Conclusion

In this exercise you learned to place different deflectors on several objects, to resize them and to edit their parameters via the **Attributes Manager**. Other types of deflectors (spheres, cylinders, etc.) are created, resized and managed in the same way.

29.11 Exercise 4 - Using Animated Textures with Particles

You can also use animated textures as particles. Animated texture particles require that animation images be embedded within a single texture. The texture is equally divided into at least as many images as you wish to display. Virtools Dev then reads the images, from left to right and top to bottom, as you will now see.

1. From the **File** menu, choose **New Composition**.
2. From **VirtoolsResources**, drag CompoundNumbers.jpg (Textures/Particles) into the **3D Layout**.
3. If necessary, in **Level Manager**, double-click CompoundNumbers (Global/Textures) to open **Texture Setup**.

   The texture is composed of four colored numbers. To create this texture, each number was created separately as a 16 x 16 pixel image, then all four

images were joined together in a graphics program to make a 32 x 32 pixel texture.

4. In **3D Layout**, click **Create 3D Frame**.
5. From **Building Blocks**, drag **SphericalParticleSystem** (Particles) onto the 3D frame in **3D Layout**. In the **Edit Parameters** dialog box, specify CompoundNumbers under **Texture**. Click **Play**.

   All four colored numbers appear at the same time. You need to tell Virtools Dev that the texture actually contains four separate images.
6. In **Schematic**, right-click **SphericalParticleSystem** and choose **Edit Parameters**. Enter 4 under **Texture Frame Count**, and click **OK**.

NOTE Another interesting parameter for this exercise is **Texture Loop** - you should leave it at No Loop for the moment but later try both Loop and To and Fro.

7. Click **Play**.

   You will see that the particles cycle through the four images. Next you will alter the **Variance** slightly, so that all images are visible at the same time.
8. In **Schematic**, right-click **SphericalParticleSystem** and choose **Edit Parameters**. Set the **Texture Speed Variance** to 400 ms, then click **OK**.

   You will see that all four numbers are now visible at the same time. Try experimenting with the other **Variances** in the **Edit Parameters** dialog box.

Once you have finished with numbers, try re-doing this exercise with the texture Compound.jpg, also found in **VirtoolsResources** (Textures/Particles). You can also create your own textures - for example you could create a texture with nine images showing differing flames, and use the particle system to create a fire effect. Patience and a good graphics program are all you need!

NOTE Animated particle textures must be arranged in a square grid format. Legal values for the number of images within an animated particle texture are 4, 9, 16, 25, etc.

29.12 Exercise 5 - Creating 3D Particles

Although simple sprites are usually enough for most particle effects, you may occasionally need other particle types. Every emitter has settings that you access by right-clicking the appropriate BB. In the **Edit Settings** dialog box, you can choose the **Maximum Number** of particles (100 is the default, and although you will rarely need more you can often do with less).

You can also choose the types of particles via the **Particle Rendering** parameter. Try out the Point, Line, Sprite and especially Fast Sprite modes yourself to see their differences. You can see a representation of the various modes available in **29-1**.

Rendering particles as **Objects** is a little different.

1. Open the file Particles05.cmo.

   You will see a 3D object, Cone, that has a very simple mesh. You can see in its **3D Object Setup** that it has only eight faces.

NOTE It is important to use simple mesh objects *only* when rendering particles as objects, otherwise your frame rate is likely to plummet.

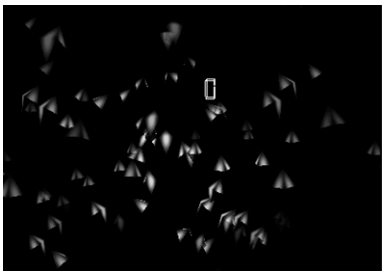
2. In **Level Manager**, right-click Cone (Global/3D Object), and choose **Place Selected in New Group**.

NOTE You must place your objects into a new group when they are to be used for a particle system. You can, of course, use more than one object for object particles.

3. Select ConeFrame (Global/Frame) and apply **SphericalParticleSystem** (Particles). Click **OK** to accept the default parameters.
4. In **Schematic**, right-click **SphericalParticleSystem** and choose **Edit Settings**. Choose Object under **Particle Rendering**, and New Group under **Objects**. Click **OK**, then **Play**.

   You will see particles composed of the 3D object (Cone) floating around the emitter, similar to **29-14**.

29-14 Rendering object particles



5. Open the BB's **Edit Parameters** dialog box, and modify **Speed** and **Size** to investigate the different visual effects you can achieve.
6. In **Level Manager,** select another simple mesh object from **VirtoolsResources** (such as those contained in 3DEntities/Primitives) and add it to your group (by dragging it to the group). Click **Reset IC** and then **Play**.

NOTE The BBs contained in Logics/Groups allow you to create particle systems with dynamic content by adding objects at run-time.

29.12.1 Conclusion

In this exercise you learned how to create particle systems composed of 3D objects. You learned that you must use a group and choose only simple 3D objects with a low polygon count. Finally, you learned that you can use the BBs in Logics/Group with object particle systems to create particle systems with dynamic content.

29.13 Multiple Particle Systems

You can create multiple sources of particles with just one emitter by *freezing* the emitter, moving it, and then reactivating it. Particles generated at each location will not change when the emitter is frozen or moved and will continue to the end of their **Lifespan**. A fireworks display using just one emitter is a simple example of this technique.

Alternatively, you can attach multiple particle system BBs to a single emitter and switch between the particle systems as necessary.

29.14 Frame Rates with Particle Systems

You now know the essentials of using particle systems in Virtools Dev. It is very easy to create stunning visual effects. However, you should not forget that particle systems use a lot of system resources, both CPU time and memory. The larger the number of particles, the greater the number of calculations required which can result in unsatisfactory frame rates. To maintain an acceptable frame rate, apply the following guidelines:

- Avoid generating a large number of particles. The default number of particles for any particle system is 100, and although you will rarely need more than this, you can very often do with much less. For example, 16 particles are all that is needed for a realistic smoke effect.
- Try to avoid having several emitters active at the same time. Especially avoid adding too many interactors or deflectors to a scene.
- Remember to hide as many parameter types as possible by clearing their check boxes in the **Edit Settings** dialog box - this will gain you some CPU time.
- Try to keep texture sizes to 16 x 16 or 32 x 32 pixels. You should also avoid rectangular textures because some graphics cards only support square textures.
- Use Fast Sprite mode (**Particle Rendering** in the **Edit Settings** dialog box) as much as possible with small particles - Fast Sprite mode uses only half the number of faces used by Sprite mode.
- Try to use the simplest deflector attributes, such as *Particle Plane Deflector* or *Particle Sphere Deflector* rather than *Particle Cylinder Deflector* and especially *Particle Object Deflector*. *Infinite Plane Deflector* is the best choice for minimizing processing power requirements.

- Only use Object mode (under **Particle Rendering** in the **Edit Settings** dialog box) with small simple objects of few faces.
- Try to place your emitters so they are not all in the camera view frustum at the same time. When a particle system is not in view, you can deactivate or freeze that particle system.

NOTE It is better to freeze a particle system, if you have not finished with that particle system, rather than stop a particle system, for it takes time for a particle stem to become stable.

Part 6 - Appendix

At the end of the User Guide, last - but certainly not least - is the Appendix. Our appendix is larger than most - mostly because of our extensive glossary. The Virtools Glossary contains definitions for many Virtools words and terms that newcomers and pros alike will come to appreciate for their clarity and valuable contextual information.

The Glossary is not all there is to the Appendix - you will also find further help for those tricky orientation and transformation questions.

The Appendix contains:

30 Glossary - until you become a pro, one of your most valuable resources; the Glossary is the definitive source for Virtools terminology

31 Controlling the Orientation of an Element - a short tutorial on controlling orientation

32 Example Transformations - more details on the matrices that underlie 3D rendering

30 GLOSSARY

30.1 How to Use the Glossary

This Glossary contains definitions for and contextual information about words, phrases, and relationships that are used in Virtools Dev.

Many of the definitions refer to one of the following fundamental terms, reproduced here for your convenience. Words that are capitalized have their own entries in the Glossary. Before you start reading the Glossary, we advise you to review the following entries:

Behavior

A description of how an element responds to the environment. Applying a Behavior makes an element interactive, either with the User or with other elements of the composition.The term Behavior is a general description and can be used in place of Building Block (BB), Behavior Graph (BG) or Script.

Behavior Building Block, BB

The fundamental interactive elements within Virtools Dev, BBs encapsulate a specific task. BBs are a visual representation of a software element known as a function.

Behavioral Object, BeObject, CKBeObject

A CKClass. Any element, within a composition, to which a Behavior can be applied.

CKClass

A generic label for any class or class definition used by Virtools Dev.

Element

A generic label for any non-behavioral (non-BB, non BG, non-Script) "thing" within a composition.

NOTE All BeObjects are Elements but not all Elements are BeObjects: neither a Parameter nor an ObjectAnimation are BeObjects yet both are Elements.

pName, Parameter Name

The name associated with a given Parameter Input (pIn), Parameter Output (pOut), Local Parameter or Setup Parameter.

pType, Parameter Type

The type associated with a given Parameter Input (pIn), Parameter Output (pOut), Local Parameter or Setup Parameter.

pValue, Parameter Value

The value associated with a given Parameter Input (pIn), Parameter Output (pOut), Local Parameter or Setup Parameter.

Type, Data Type

A representation of a specific genre of information and the rules for determining the validity of the information. A type can be a CKClass. For example, one of the most basic data types in Virtools Dev is *integer*. An integer is any whole number in the range $-(2^{31})$ through $+(2^{31} - 1)$. Therefore, the integer data type defines a genre of information: whole numbers and the rules for the range of legal values.

A 3D Entity is a much more complex data type and represents an object instantiated from the class CK3dEntity.

30.2 Definitions and CKClasses

A number of the entries in this glossary are for the classes within the Virtools

Dev SDK. The headings for these entries are of the form:

2D Entity, CK2dEntity

The first term in the entry, 2D Entity, is the form used as a pType. The second term in the entry, CK2dEntity, is the formal name of the corresponding class within the Virtools Dev SDK. The formal name always starts with the letters "CK".

30.3 Terms and Definitions

2D Curve, CK2dCurve

A CKClass. A graphical description of the relationship between two values. For example, in a **Bezier Progression** (Logics/Loops) you could use a pIn in the form of a 2D Curve to show acceleration over time.

2D Entity, CK2dEntity

A CKClass. An element with a position and scale in the screen's 2D coordinate system.

NOTE The CKClasses 2D Sprite (CKSprite) and Sprite Text (CKSpriteText) descend from this Class.

2D Frame

An instance of CK2dEntity. A 2D Frame may have a Material and a Texture. A 2D Frame is typically used as a place holder when creating a user interface.

2D Sprite, CKSprite

A CKClass. A 2D image rendered in the background (behind all 3D elements) or in the foreground (in front of all 3D elements), a 2D Sprite can be of arbitrary size.

2D Sprites are often used to create interface elements. However, whenever possible, use a 2D Frame (with a material and texture) for 2D Frames require less processing power than 2D Sprites.

3D Entity, CK3DEntity

A CKClass. An element in Virtools Dev that has a position, orientation and scale in 3D space.

3D Frame

An instance of the class CK3dEntity. A 3D Frame has a position, orientation, and scale. A 3D Frame does not have an Author controlled Mesh. When visible, a 3D Frame is represented as a 3D cross.

A 3D Frame is often used as a reference point. For example, placing a camera relative to the 3D Frame's position or orientation.

3D Object, CK3dObject

A CKClass. An element in Virtools Dev that has the characteristics of a 3D Entity and a Mesh.

3D Sprite, CKSprite3D

A CKClass. An element composed of a 2D image that is rendered in 3D space. A 3D Sprite can have a position, orientation, and scale.

Absolute Coordinates

In 2D, a screen position expressed in pixels.

In 3D, a position expressed in the world coordinate system.

Ambient Color

See "Material Color" on page 201.

Ambient Light

A pName. The non-directional, background light in a composition.

Ancestor, Parent

Any element, within a hierarchical relationship, that has one or more Child elements (children) or Descendents. Strictly, a Parent refers to the immediate Ancestor only.

Angle

A pType. A value expressed in the form **Turns:Angle** where **Turns** is the number of complete rotations and **Angle** is expressed in degrees.

NOTE CK2 performs <Angle> calculations using radians and not degrees. There are 2*PI radians in 360 degrees. An Angle of 0:180, expressed as a Float, has a value of PI (approximately 3.14159).

NOTE CK2 considers the pTypes <Percentage>, <Float> and <Angle> as being the same type.

Animation

A pName. A process by which an element is modified in 3D space, making the element appear dynamic (rather than static). An Animation is composed of Keyframes, where each Keyframe has a position, scale and orientation.

Animation Frame, Keyframe

A pName. A Keyframe is a reference frame in an animation.

For example, a Keyframe in a Character Animation is typically a pose that captures the state of the character at a critical instant in an Animation. Keyframes are used to reduce the data storage requirement for an Animation. Intermediate Animation Frames (those frames required to maintain a smooth Animation between Keyframes) are constructed by a process of Interpolation.

Animation Step

A pName. The current position within an Animation, expressed as a Percentage.

Applying a Building Block to an Element

Once attached, BBs are *applied* to a an element either

- *explicitly* - if the BB is targetable *and* the Author has defined the target, or

- *implicitly* - applied to the owner element, the element to which the Script containing the BB is attached.

Array, CKDataArray

A CKClass. A rectangular arrangement of cells, created as rows and columns, where the columns define the types of data stored in the array and where the rows contain values. Each cell contains a distinct data element. The data types supported by Arrays include:

- Integer
- Float
- String
- CKObject
- Parameters

Attaching a Building Block to an Element

BBs are *attached* to an element at the moment of mouse button release in a drag and drop operation.

Attribute

A Parameter associated with a BeObject. An Attribute has a Name and may also have a Category. An Attribute typically consists of one or more Author-editable Parameters. However, some Attributes, such as ZBuffer Only, cannot be edited by the Author.

For example, an element with the Floor attribute is identified to the Behavioral Engine as an element that should be treated as a floor so that Characters walk *on* the element and not *through* the element.

Virtools Dev includes numerous predefined Attributes. Authors can also define their own Attributes.

Author

A person using Virtools Dev to create interactive content or an application.

Author Mode

A state within Virtools Dev; Author Mode occurs when a composition is not playing, and the Behavioral Engine is not active.

Axis

A pName. A vector used to define a coordinate system.

Behavior

A description of how an element responds to the environment. Applying a Behavior makes an element interactive, either with the User or with other elements of the composition.

The term Behavior is a general description and can be used in place of Building Block (BB), Behavior Graph (BG), or Script.

Behavior Building Block, BB

The fundamental interactive elements within Virtools Dev, BBs encapsulate a specific task. BBs are a visual representation of a software element known as a function.

BBs can be categorized as:

1. Single Action: the BB completes processing within the current Frame.

   A Single Action BB can stand alone or be part of a Behavior Loop. Example: **Set Fog** (World Environments/Global).

2. Internally Looped: the BB is turned *On* and the BB is activated every Frame until the BB is turned *Off*. Example: **Keyboard Controller** (Controllers/Keyboard).

3. Externally Looped: the BB completes one step in the BB's process loop within the current Frame. Example: **Bezier Progression** (Logics/Loops).

   If the Author wants the BB to operate in the typical manner, an external activation feedback loop is required.

NOTE An externally looped BB does *not* require that an external activation feedback loop be present. It is possible to construct a Script in a manner such that the external activation feedback loop is not required.

BBs can have one or more of

1. a Custom dialog box, denoted by a C in the lower left hand corner of the icon
2. internal Settings, denoted by an S in the lower left hand corner of the icon, or
3. a Variable configuration, denoted by a V in the lower left hand corner of the icon, that may include
   - a variable number of number of bIns
   - a variable number of number of bOuts
   - a variable number of number of pIns
   - a variable number of number of pOuts
   - the ability to change the type of one or more of the pIns and pOuts
4. The ability to send a message, denoted by a *Sender* icon in the lower left corner of the BB
5. The ability to receive a message, denoted by a *Receiver* icon in the lower left corner of the BB

The following terms are used when describing how BBs are processed.

1. *Activate* denotes the first time a BB is executed in a Frame.
2. *Deactivate* denotes that an internally looped BB is no longer active.

3. *Trigger* denotes that a BB is activated by the external loop feedback path.
4. *Receive* denotes that some BBs can be activated on more than one bIn per Frame; we say that the BB *receives* activations on the relevant pIns. Example: **LIFO** and **FIFO** (both in Logics/Streaming).
5. *Store* denotes that some BBs keep a record of received activations; we say that the BB *stores* activations. The BB then processes the record of received activations and generates new activations according to the rules of that BB. Example: **LIFO** and **FIFO**.
6. *Generate* denotes that some BBs generate more activations than the BB receives. Example: **All But One** (Logics/Streaming).

Behavior Graph, BG

A graph composed of one or more of the following elements: BBs, Parameter Operations, Parameters, Behavior Links, Parameter Links, Comments, Shortcuts, other BGs, etc. In other words, a BG is a visual representation of a Behavior.

At first glance, a BG can look almost exactly like a Script. However, a BG is distinct from a Script because the Author of the BG deliberately encapsulated the Behavior. The Author encapsulated and named the BG so that the BG can be saved and reused.

Virtools Dev treats a BG exactly the same as a BB. A BG can be attached to an element and applied to an element in the same manner as a BB.

A BG can be considered an Author defined BB that, to an Author, works in exactly the same way as a BB – a BG can have pIns, pOuts, bIns and bOuts.

NOTE Any time the documentation for Virtools Dev refers to using a (Behavior) Building Block or a BB within a composition, the reader can substitute Behavior Graph or BG.

A BG can be constructed by encapsulating a portion of a Script or all of a Script. A BG can also be constructed as an empty BG and elements added until the desired functionality is achieved.

A BG can be *expanded* to show the elements that make up the BG or *collapsed* to hide the elements that make up the BG.

An expanded BG is recognized by the surrounding bounding box and the three diagonal lines in the bottom right corner of the bounding box.

A collapsed BG looks like a BB with the following exceptions: the BG label is in Bold face and the type is Dark Gray in color.

Behavior Input, bIn

Located on the left side of a BB/BG. Some part of the code within a BB is executed when a bIn is *activated, deactivated,* or *triggered*. Some BBs (particularly Logics/Streaming) can also *receive* and *store activations*. For example, **FIFO** and **LIFO** can receive *n* activations per Frame but both can only generate one activation per Frame.

Behavior Output, bOut

Located on the right side of a BB/BG. Generally, a bOut *activates* when the processing to be performed by a BB in the current Frame (process loop) is complete. Some BBs (particularly Logics/Streaming) can also *generate activations* at their behavior outputs.

For example, **Keep Activated** and **All But One** both generate activations (activate more bOuts than bIns).

See Behavior Loop for further description of the processing done by a BB in a given Frame.

Behavior Link, bLink

A link within a Script that propagates activations from bOuts to bIns. A behavior link can have a Behavior Link Delay.

(Behavior) Link Delay

A bLink can have an associated delay, where the Link Delay is measured in Frames (process loops). The Link Delay can be:

- 0 – propagate the activation within the current Frame
- 1 – propagate the activation in the next Frame
- n - propagate the activation in the n^{th} Frame after the current Frame

Behavior Loop

A series of BBs (and/or BGs) connected by bLinks. BBs are connected together in a manner that causes the BBs to be activated or triggered in a repetitive fashion.

In other words, a Behavior Loop is the visual representation of a repetitive operation. A repetitive operation is also known as an Iteration and the process of repeating an operation is also known as Iterating.

In any implementation of a repetitive operation, a significant concern is to ensure that there is an acceptable upper limit to the number of times that the operation repeats. In other words, there needs to be a mechanism to ensure that an operation is not repeated indefinitely.

Virtools Dev allows the Author to define the maximum number of operations that may be performed in a single Frame via the *Max Behavioral Iterations* setting in the Schematic.

NOTE The default value for Max Behavioral Iterations is 8000. The value of Max Behavioral Iterations is saved with the composition.

Any Behavior Loop may have a cumulative delay of zero, as long as the loop is not constantly active and as long as the number of iterations in a Frame does not exceed the value of Max Behavioral Iterations. For example, Array and Group Iterators (Logics/Arrays and Logics/Groups) often have a loop delay of 0 so that all elements within the Array or Group are operated on within a single Frame. However, the loop delay can be any positive value in the range of 0 to 32767.

NOTE Looping BBs (such as **Timer** (Logics/Loops)) that are time-based require a cumulative loop delay of 1 Frame or the BB automatically changes to Frame-based processing.

Behavioral Object, BeObject, CKBeObject

A CKClass. Any element, within a composition, to which a Behavior can be applied.

Bezier (Curve)

A pType. A Bezier Curve (see **Bezier Progression** (Logics/Loops)) is a type of a 2D Curve characterized by smooth transitions between curve segments (as compared to Linear (see **Linear Progression** (Logics/Loops)).

Billboard

A pValue. An orientation constraint applied to an element. An element with a Billboard constraint always faces the current viewpoint.

BodyPart, CKBodyPart

A CKClass. One or more 3D Entities combined to form a Character.

Boolean

A pType. A data type with only two permitted values, represented as a check box. The check box is either selected (TRUE) or not selected (FALSE).

Bounding Box

The smallest box or rectangular volume that encloses an element or set of elements. A bounding box is typically used in performing collision detection.

Camera, CKCamera

A CKClass. An element that provides a viewpoint from which a composition is rendered.

NOTE A Target Camera (CKTargetCamera) descends from this Class. A Target Camera is a Camera that always points towards its Target.

Cell

A data point within an Array. For example, an Array with two rows and three columns contains six cells.

Character, CKCharacter

A CKClass. A 3D Entity composed of one or more BodyParts, arranged as a Hierarchy. A Character typically has Animations.

Channel

A pName. A Material added to a Mesh that blends with other Materials to create a visual effect. Virtools Dev supports a maximum of 10 Channels per Mesh.

Child, Descendent

Any element, within a hierarchical relationship, that has a Parent (Ancestor) element. Strictly, a Child refers to the immediate Descendent only.

CKClass

A generic label for any class or class definition used by Virtools Dev.

Class, Class Definition

A template for a classification system; a description of a type of element. From object oriented design, a design technique used to group elements based on common characteristics and common behaviors.

Class Hierarchy

A classification system whereby classes are related via parent – child relationships. A parent class is more general than a child class. A child class is a specialization of a parent class and inherits the characteristics and behaviors of the parent class unless specifically redefined in the child class.

CK2

The Virtools Dev Behavioral Engine, responsible for the execution of all behaviors within a composition.

CK_ID

A unique identifier assigned to every element in a composition. The CK_ID is assigned as the element is added to a composition. The CK_ID for a given element is regenerated every time a composition is loaded and is not guaranteed to remain constant between reloads.

CMO – Virtools Composition File

An arrangement of one or more elements and associated behaviors, assembled within Virtools Dev. A composition contains a Level and, if defined by the Author, may contain Scenes and Places.

Curve, CKCurve

A CKClass. An element formed by a set of Curve Points that define a series of line segments connected to create a single line.

Curves are **Open** (the Curve forms a line) or **Closed** (the Curve forms a loop). Curves and curve sections are **Linear** (composed of straight line segments) or **Spline** (composed of smoothly varying curves).

The Curve Points of the curve lie on the curve. The shape of the curve is changed by moving or editing the Curve Points.

Curves are 3D elements and can be made visible or invisible in Author Mode and Player Mode.

Curve Point, CKCurvePoint

A CKClass. A 3D Entity used to define a Curve.

Custom Dialog Box

A C in the bottom left corner of a BB indicates that a special dialog box is used to control how the BB functions. Example: **Unlimited Controller** (Characters/Movement), **Keyboard Mapper** (Controllers/Keyboard).

Default Orientation

The default relationship between an element and the world coordinate system. The Default orientation assumes that the element faces in the direction of the positive Z-Axis (**Dir**), that the top of the element is in the direction of the positive Y-Axis (**Up**) and that the right side of the element is in the direction of the positive X-Axis (**Right**).

Dependency (Options)

A pName. How a copy or delete operation affects the CKObjects used by (*aggregated with*) that element.

For example, when deleting an element, the Author can choose to

- delete the element only (**No Dependencies**)
- delete the element, and all elements used by that element (**Full Dependencies**)
- delete the element, and only selected elements used by that element (**Custom Dependencies**)

Descendent, Child

Any element, within a hierarchical relationship, that has a Parent (Ancestor) element. Strictly, a Child refers to the immediate Descendent only.

Dest, Destination

A pName. From Material, sets a surface's destination blending coefficient.

From Transformation, the end point of a translation. The Destination can be a coordinate (expressed as a <Vector>) or an element within the composition.

Diffuse (Color)

See "Material Color" on page 201.

Diffuse (Light)

A pName. The directional light in a composition.

Dir

A pName. The facing direction of an element, generally the positive Z-axis.

Element

A generic label for any non-behavioral (non-BB, non BG, non-Script) "thing" within a composition.

NOTE All BeObjects are Elements but not all Elements are BeObjects: neither a Parameter nor an ObjectAnimation are BeObjects yet both are Elements.

Emissive (Color)

See "Material Color" on page 201.

Extents

A pName. The bounds or limits of an element, typically as projected onto another element or onto the planes defined by the world coordinate system.

Face

A triangular geometric element formed by three Vertices, a part of a Mesh.

Flow (Activation)

The order in which the BBs within a Script are processed. The activation flow follows a path through the graph formed by BBs and bLinks.

Frame, Process Loop

A single pass through the processing of all elements of the composition. Simplistically, Virtools Dev processes all behaviors then renders the scene for each pass through the process loop (Frame). Each pass through the process loop creates and displays a new image.

NOTE Advanced Authors often refer to the process loop as a *Frame* or a *rendering Frame*. Do not confuse a *Frame* or a *rendering Frame* with a *2D Frame,* a *3D Frame,* or an *Animation Frame*.

Grid, CKGrid

A CKClass. An element that divides the volume enclosed by the Grid into a 2D array of rectangular volumes known as *squares.* A Grid provides a mechanism for projecting an element's position in 3D coordinates to a 2D coordinate system.

Group, CKGroup

A CKClass. A collection of references to elements, used to create a logical relationship between the members of the Group.

Guide

In Author mode, a rectangular grid used to guide placement of elements.

Hierarchy

From object oriented design, a logical organization that enforces a parent-child organization on arbitrary elements.

For example, the file system on a computer is a hierarchical organization where each drive is organized as a hierarchy with the drive letter as the *root* of a given hierarchy. Organization proceeds in a tree-like structure where a drive can contain folders and files and where folders can contain further folders or files but files can not contain other files. Any folder is the *parent* to all folders and files within the folder. A folder within another folder is a *child* of the encompassing folder. Therefore, a folder can be both a parent to other folders

and files and a child of another folder. A folder can have many children but a folder can only have one parent.

Note that root is a relative term; any parent is also the root for all of that parent's child elements within a hierarchy.

Within Virtools Dev, a Character is the most common form of a hierarchical element. The various elements that form the body (such as torso, legs, arms, etc.) are arranged in a hierarchical manner.

Hierarchies can be explored, created and altered using the **Hierarchy Manager** or BBs.

Hierarchy (pIn)

A pName. A Boolean pIn in the form of a check box – if the check box is selected, then the BB affects the elements' children.

Homogeneous Coordinates

In 2D, a screen position expressed relative to the current screen resolution along each axis. Homogeneous Coordinates are used to ensure that, no matter the screen resolution, 2D elements can maintain a constant relative size.

Homogeneous Coordinates are limited to the range 0.0 to 1.0.

Initial Conditions (IC)

On an element, a snapshot of the current state of an element; a record of the data internal to an element at the instant that the Initial Conditions are set.

On a Script, a record of the current values of the local Parameters and the current values of the input Parameters at the instant that the Initial Conditions are set.

Instance, Instantiation

An Object is an instance of a Class; an Element that complies with or was created from a Class Definition. An Object is Instantiated from a Class Definition.

Interpolator, Interpolate

A BB that interpolates between two values, that calculates a value between the two values.

Interpolation determines the value at a given Percentage of the way between the first value and the second value.

Iterator, Iterate

A BB that Iterates from a first value to a second value; for example: **Counter** (Logics/Loops). Iteration counts from a first value to a second value using a given increment per iteration.

A BB that Iterates over all members of a Collection; for example: **Collection Iterator** (Logics/Loops).

Keyframe, Animation Frame

See "Animation Frame, Keyframe" on page 186.

Layer, CKLayer

A CKClass. On a Grid, a set of values associated with each square of the Grid. Multiple Layers can be associated with a single Grid.

Light, CKLight

A CKClass. An element used to provide real-time lighting as opposed to using pre-lit lighting.

NOTE A Target Light (CKTargetLight) descends from this Class. A Target Light is a Light that always points toward its Target.

Level, CKLevel

A CKClass. The root element for a composition; the ancestor for all elements in a composition.

Local Parameter

A Parameter contained within a Script, represented by a small rectangle. Local Parameters are either located above pIns (the small triangles on top of a BB/BG/paramOp) or located by themselves within a Script. Local Parameters are connected to pIns by Parameter Links.

NOTE Two unique Local Parameters are allowed to have the same name. However, this practice is not recommended due to the potential for confusion. For instance, when you copy and paste a Local Parameter, you copy the Parameter's name, the Parameter's type and the Parameter's value to a new local Parameter: these two Parameters are different, even though they have the same name (due to the copy and paste operation). The Author is encouraged to use unique names for each Parameter that they create.

Virtools Dev maintains a unique identifier for each local Parameter. To view a local Parameter's identifier (called the CK_ID), select "CK Properties" in the local Parameter's context menu.

Manager

A Plugin that performs system wide tasks such as collision management.

Material, CKMaterial

A CKClass. The surface characteristics of an element, how light affects the element. A Material often has a Texture.

Material Color

The perceived color of an element, defined by the way the element's Material interacts with the scene lighting. The Material Color has four components:

1. **Ambient Color**: describes how a Material reflects the ambient light in a scene. Ambient light and ambient reflection are non-directional.

Typically, the ambient light level in a scene is much lower than the diffuse light level. Therefore, ambient reflection typically has a lesser impact on the perceived color of a Material than diffuse reflection and is most noticeable when little or no diffuse light reflects off the Material.

2. **Diffuse Color**: describes how a Material reflects the diffuse light in a scene. Diffuse reflection is directional - the angle of incidence of the diffuse light affects the overall intensity of the reflection.

   Typically, the diffuse light is the dominant light source in a scene. Therefore, diffuse reflection typically plays the largest part in determining the perceived color of a Material.

NOTE The alpha value of the diffuse color is used for alpha transparency calculations.

3. **Specular Color**: describes how a Material reflects the specular light in a scene. Specular lighting is responsible for the creation of specular highlights on a Material.

   The appearance of a highlight is dependent on the viewing angle and the specular power. The greater the value of the specular power, the thinner the specular highlight.

4. **Emissive Color**: describes how a Material can be used to make a rendered object appear to be self-luminous. A Material's emissive color creates the illusion that a Material is illuminated (from within), without incurring the computational overhead of adding a light to the scene.

NOTE A self-illuminated Material does not act as a source of illumination for other elements.

Matrix

An ordered collection of data. The local matrix and the world matrix associated with each 3D Entity are used to store the position, orientation, and scale of that element.

Mesh, CKMesh

A CKClass. A collection of faces that define the geometric surface of an element. A Mesh is usually covered by a Material.

Message, CKMessage

A CKClass. A means of transferring information between elements or between Scripts. Typically, messages are used to signal a change in state, to request that some task be performed, and to signal that some task has completed.

NOTE Messages always have a delay of one pass through the process loop. That is, a message is sent in the current Frame but the message is not received until the next Frame (process loop). In other words, messages experience a one Frame delay before they are delivered.

NMS – Virtools Script File

A file containing a Script or a BG.

NMO – Virtools Object File

A file containing one or more elements, with or without their applied Scripts.

Nodal Link

A link between Nodes in a Nodal Path. A Nodal Link may allow movement in one direction only or in both directions. A Nodal Link also has a *difficulty* value used to determine the cost of following that Nodal Link.

Nodal Path, Path

A map, composed of Nodes and Nodal Links between Nodes, used to predetermine the set of possible paths for an element between a source and a destination. Often used to determine the shortest allowed path between two points.

Node

An endpoint for a Nodal Link, a junction between links in a Nodal Path.

Normalization, Normalized Value

A value constrained to the range 0.0 through 1.0 by scaling relative to a maximum value. A value is normalized by dividing the value by that value's maximum value.

Object

From object oriented design, an instance of a class; an element rather than a description of an element (a class).

Obstacle

An element tested by collision detection mechanisms.

Orientation

The correlation between an element's Right, Dir, and Up vectors, an element's local coordinate system, and the world coordinate system.

Origin

The point at which the vectors that define a coordinate system intersect.

Parameter

Used to transfer data values between behaviors and to add information to BeObjects (through Attributes). Many behaviors use the values of their pIns to control their processing.

A Parameter has a name (pName) and contains a value (pValue) expressed in a given type (pType) such as <Integer>, <String>, <Vector>, etc. A Parameter can be static (the value is fixed, such as a value entered by the Author) or dynamic (the value can change, based upon other factors). The value of a Parameter is provided to a Behavior or to a paramOp via a pLink or via a shortcut.

Parameter Input, pIn

Data values received by a paramOp or by a BB. pIns have a *source*. pIns are represented by the small triangles on top of a BB/BG/paramOp.

In traditional programming terms, Parameter Inputs are the arguments to the function encapsulated by a BB, BG or paramOp.

Parameter Link, pLink

A link that propagates a Parameter value either from a pOut or from a Local Parameter to a pIn.

NOTE When a BB is added to a Script (in the Schematic) or to an element (in the 3D Layout), the links between the default Local Parameters and the corresponding pIns are not visible due to the scale of the image. However, the link is immediately visible if the Local Parameter is moved from the default location.

Parameter Operation, paramOp

A simple operation performed on a single Parameter or performed between a pair of Parameters. A paramOp is only evaluated when the result is requested by another paramOp or a BB.

Many calculations and data retrieval operations (**Get Value**) are implemented as paramOps.

Parameter Operation Link

A simple operation, automatically performed, on a single Parameter when a pOut is connected to a pIn of incompatible type and at least one paramOp exists to convert the pOut type to the pIn type. A Parameter Operation Link is a pLink that performs a type conversion.

The type conversion occurs automatically and the default paramOp icon is *not* displayed. Instead, a Parameter Operation Link is identified by the name of the paramOp displayed along the link. By default, a Parameter Operation Link is displayed in a different color than a pLink.

Parameter Output, pOut

Data values generated by a paramOp or by a BB. pOuts have a *destination*. pOuts are represented by the small triangles on the bottom of a BB/BG/paramOp.

In traditional programming terms, pOuts are the values returned by the function encapsulated by a BB, BG or paramOp.

Parent, Ancestor

Any element, within a hierarchical relationship, that has one or more child elements (children) or Descendents. Strictly, a Parent refers to the immediate Ancestor only.

Path, Nodal Path

A graph composed of Nodes and Nodal Links between Nodes, used to predetermine the set of possible paths for an element between a source and a destination. Often used to determine the shortest allowed path between two points.

Percentage

A pType. A value bound to the range `0.0` to `1.0`, a value expressed as a percent in the range of `0%` to `100%`.

NOTE CK2 considers the pTypes Percentage, Float and Angle as being the same type.

Place, CKPlace

A CKClass. An abstract element used to define the elements found within a physical locale in a composition. A Place is a geometric construct used to define an area of related geometry.

For example, within a building, each room can be defined as a separate Place.

Places contain the elements within the Place. An element in a Place does not appear in the Global list of elements.

Places are very useful when using portal optimization to reduce the processing requirements of a composition.

Player

An application, such as the Virtools Web Player, that can process and render a Composition.

Player Mode

A state within Virtools Dev; Virtools Dev enters Player Mode when a composition is playing and rendered in 3D Layout.

Plugin

A Dynamic Link Library (DLL) that extends the capabilities of the base CK2 engine.

pName, Parameter Name

The name associated with a given Parameter Input (pIn), Parameter Output (pOut), Local Parameter or Setup Parameter.

Portal

A connector between two Places, used by the Portal Manager to reduce the render processing requirements of a composition. A Portal is a special 3D Entity (a 3D Frame with a Portal flag) that is used to determine which Places are potentially visible from the current viewpoint and, therefore, rendered.

Priority

Allows the Author to control the activation order of elements and behaviors in the current Frame.

By default, Virtools Dev implements a pure message passing model where each object acts independently and asynchronously. In a pure message passing model, there is no priority – each object exhibits the appropriate behavioral

response to the received messages. However, there may be times when an Author must be able to guarantee the order in which behaviors are activated. Virtools Dev allows the Author to specify the activation order by controlling the priority of objects, scripts, BGs and BBs.

The priority for an object or behavior can be set to a value in the range +32767 to –32768, where +32767 is the highest priority.

Behavior processing is performed in order of priority. The objects within a composition that have scripts attached are sorted from highest priority to lowest priority. The object with the highest priority is processed first.

If there are multiple Scripts attached to that object, the Script with the highest priority is processed first. All of an object's scripts are processed before proceeding to the next highest priority element.

Within a Script, if several BBs are attached to the Start (not recommended) and all BBs have the same link delay, then the BB with the highest priority is processed first.

If several elements have the same priority, then it is impossible to predict which element will be processed first – the Behavioral Engine will simply choose one at random.

Controlling priority can be useful for resolving conflict when BBs, BGs, or Scripts seem to be interfering with each other.

Process Loop, Frame

A single pass through the processing of all elements of the composition. In the simplest model of the process loop, Virtools Dev executes all behaviors then renders the scene. More detailed models are available in the Virtools Dev SDK documentation.

NOTE Advanced Authors often refer to the process loop as a *Frame* or a *rendering Frame*. Do not confuse a *Frame* or a *rendering Frame* with a *2D Frame,* a *3D Frame,* or an *Animation Frame*.

Progression

A BB that iterates from a first value to a second value.

A Progression counts from a first value to a second value using a given increment per iteration. The increment can be fixed (**Linear Progression** (Logics/Loops)) or variable (**Bezier Progression** (Logics/Loops)).

Progression Curve

A pName. A mechanism for visually describing the increment used at each step of a Progression.

pType, Parameter Type

The type associated with a given Parameter Input (pIn), Parameter Output (pOut), Local Parameter or Setup Parameter. A pType is expressed as <pType>.

pValue, Parameter Value

The value associated with a given Parameter Input, Parameter Output, Local Parameter or Setup Parameter.

Referential

A pName. The element used as the point of reference for an operation. Typically, the element used as the point of reference for calculating a Transformation (for example, a translation).

Render Object, CKRenderObject

A CKClass. An element that is rendered, that is shown on-screen in the render window.

Resource

Any element, BB, or BG, that can be directly imported into a composition (via drag and drop or file selection); any element in a Virtools Dev compatible file format.

Right

If an element is assumed to be a Character, the direction of the element's right hand with respect to the origin of the Local Coordinate System.

Root

The topmost element within a hierarchy. Root is also a relative term; any parent is also the root for all descendent elements within a hierarchy.

The Level is the root of a composition.

Scale

A pName. A factor that describes the element's current size compared to it's original size.

Scene, CKScene

A CKClass. An abstract element used to define the elements within a narrative unit of a composition. A Scene is a logical construct used to define a group of related elements and is not restricted to a geometric locale within a composition.

For example, scenes can be used to control *which* elements are active *at what time*: in *this* part of the story, only *these* elements are active whereas in *that* part of the story only *those* elements are active. Remember, only active elements are processed.

Scenes contain references to elements and not copies of elements or the elements themselves.

Scenes are very useful when attempting to reduce the processing requirements of a composition.

Script

The visual representation of a behavior, *applied* to an element, as represented in the Schematic.

A Script is composed of two parts – a header and a body.

The Script header displays the name and owner of the Script, and optionally a small snapshot.

The Script body is composed of the Start and one or more BBs, BGs, paramOps, Parameters, bLinks, pLinks, comments, etc.

Mathematically, a Script is a graph whose nodes represent operations (BBs or paramOps) and whose edges represent possible paths of data flow (pLinks) and program flow (bLinks).

A Script is processed by following the bLinks as directed by the BBs contained within the Script.

Setting

An S in the bottom left corner of a BB indicates that the BB has internal settings that can be edited through its' context menu.

Settings can take many forms. For example, Settings can control whether a BB is time-based or Frame-based (**Linear Progression** (Logics/Loops)), or what aspects of a BB are actually calculated at each activation (the Settings for **Mouse Waiter** (Controllers/Mouse) control what bOuts are available and, therefore, calculated each Frame).

Setup Parameter

A parameter available in the Setup for an element.

Sibling (Classes)

Classes that share a common parent class.

Sound, CKSound

A CKClass. A data set containing a digital representation of a sound.

NOTE Midi Sound (CKMidiSound) and Wave Sound (CKWaveSound) descend from this Class.

(Parameter) Shortcut

A Parameter Shortcut is similar to the desktop shortcuts used in Windows. A Parameter Shortcut is composed of a destination and a source, each identified by arrow icons.

A Shortcut is actually another instance of a particular Parameter. Therefore, the CK_ID of the destination has the same CK_ID as the source.

A Shortcut is dynamically updated and is often used to make a Parameter available across Script boundaries (the source is in one Script and the destination is in another Script) or used to make a Parameter available within a Script without using a pLink. A Shortcut can also be used to make a Parameter available to multiple locations without requiring a pLink between the source and each destination.

NOTE An Attribute is a Parameter attached to a BeObject. Therefore, a shortcut to an Attribute can also be created.

Specular

See "Material Color" on page 201.

Specular (Light)

A pName. A directional light with the specular flag set, a light responsible for highlights on a Material.

Sprite Text, CKSpriteText

A CKClass. A Sprite used to draw text on the screen.

Src, Source

A pName. From Material, sets a surface's blending coefficient.

From Transformation, the start point of a translation. The Source can be a coordinate (expressed as a <Vector>) or the identity of an element within the composition.

Start

The left-most icon within the body of a Script, the point at which a Script begins.

Target Parameter

A special type of pIn used to explicitly identify the element affected by the Behavior.

When a Behavior is attached to an element, that element becomes the *owner* of the Behavior. Typically, a Behavior attached to an element is *implicitly targeted* at the Script owner.

For example, Translate (3D Transformations/Basic) normally modifies the position of its owner.

However, an Author may want a Behavior to affect a different element than the owner. In this case, the Behavior must be *explicitly targeted* at a different element.

Alternatively, an Author may attach a Behavior to an element of a different type than the type supported by the Behavior (for example, an Author can attach a Rotate behavior to a texture). A Target Parameter is automatically generated by Virtools Dev in the case of an incompatible class.

A BB is *targetable* only if there is a "T" in the Targetable column (between the "Apply to" and "Description" columns) within the Building Blocks window.

If a Target Parameter does not already exist on a targetable behavior, a Target Parameter can be added by selecting "Add Target Parameter" from the context menu. A new pIn is created as the leftmost pIn. A Target Parameter input is

identified by a pair of small squares (rather than the usual single small triangle for a regular pIn).

See also *Applying Building Blocks to Elements*.

Texture, CKTexture

A CKClass. An image used by a Material to give an element a certain appearance. A Texture should have dimensions that are an integer power of 2 (e.g. 32 * 64, 128 * 128).

Transformation

An operation that affects one or more of the position, orientation, and scale of an element.

Type, Data Type

A representation of a specific genre of information and the rules for determining the validity of the information. A type can be a CKClass.

For example, one of the most basic data types in Virtools Dev is *integer*. An integer is any whole number in the range $-(2^{31})$ through $+(2^{31} - 1)$. Therefore, the *integer* data type defines a genre of information: whole numbers and the rules for the range of legal values.

A 3D Entity is a much more complex data type and represents an object instantiated from the class CK3dEntity.

Up

The Y-Axis in the default orientation.

User

A person interacting with a CMO or with an application created using Virtools Dev; the end user.

UV (Texture Coordinates)

The coordinate system used to define texture coordinates on a Mesh.

U and V coordinates are normalized to the size of the texture. A UV coordinate of (0,0) indicates the top left corner of the texture. A UV coordinate of (1,1) indicates the bottom right corner of the texture.

Variable

A V in the bottom left corner of a BB indicates that the Behavior is variable. That is, the Author can do one or more of the following:

1. add bIns,
2. add bOuts,
3. add pIns,
4. add pOuts,
5. change some or all of the types of the pIns and/or pOuts.

For example, **Sequencer** (Logics/Streaming) supports the construction of additional bOuts via the context menu. **Parameter Selector** (Logics/Streaming) supports the construction of additional pIns via the context menu.

NOTE When changing pTypes, not all pTypes are available for all BBs.

For example, **Calculator** (Logics/Calculator) can accept pIns of pType <Float> or pType <Vector> only, whereas **Identity** (Logics/Calculator) can accept any pType. The Author must check each behavior's documentation to determine the supported data types.

Vector

A pType. A directed quantity; a magnitude and a direction. A Vector is always defined relative to a Referential.

Virtools Dev uses two types of vectors: a Vector2D for 2D coordinate systems (for example, screen coordinates expressed as [X,Y]) and a Vector for 3D coordinate systems (for example, world coordinates expressed as [X,Y,Z]).

The magnitude of a vector (also known as the length of a vector) is defined as:

1. Vector2D

   Magnitude = SquareRoot($X^2 + Y^2$)

2. Vector

   Magnitude = SquareRoot($X^2 + Y^2 + Z^2$)

The magnitude of a vector can be calculated using the paramOp **Get Magnitude**:

1. Vector2D

   <Float> **Get Magnitude** <Vector2D>

2. Vector

   <Float> **Get Magnitude** <Vector>

The direction of a Vector is defined as a ray (a directed line) with the origin of the ray at the origin of the Referential and extending through the point (relative to the origin of the Referential) defined by the value of the Vector.

Vertex

A point in 3D space, used to define a Face in a Mesh. A Face is defined with three vertices.

VMO - Virtools Player File

A composition that has been exported from Virtools Dev using the Export to Player command from the File menu. The contents of a VMO file are hidden and can not be opened or edited by Virtools Dev.

Workset

An arbitrary collection of elements and scripts constructed in the Level Manager. A Workset is used to organize a composition (without regard to the type of an element) and to control the visibility (within the Schematic) of the Scripts attached to members of the Workset.

31 CONTROLLING THE ORIENTATION OF AN ELEMENT

Virtools Dev can control the orientation of an element in two ways:

1. temporarily (until the next time the orientation is deliberately changed), via **Set Orientation** (3D Transformation/Basic), and
2. permanently (within the composition) with **Change Referential** (Mesh Modifications/Local Deformation).

Set Orientation calculates a transform (a transformation matrix) that applies the appropriate operations to transform the element's local coordinate system into the desired coordinate system. The transform is applied to the desired element every time **Set Orientation** is activated.

Set Orientation is often used to keep an element facing in the same direction as the referential, particularly if the referential changes often. Virtools Dev can maintain a copy of the element's original orientation (via Initial Conditions) if you need to be able to restore the element to its original orientation.

Change Referential also calculates a transform (a transformation matrix) that applies the appropriate operations to transform the element's local coordinate system into the desired coordinate system. The transform is applied to the desired element when **Change Referential** is activated and, within the composition, *the change is permanent*. You *must* ensure that no Initial Conditions (IC) are set on an element before applying **Change Referential**.

The **Change Referential** BB is often used to ensure that a group of elements share a common orientation.

Set Orientation, due to its dynamic nature, is more flexible than **Change Referential**. However, the increased flexibility comes at the cost of additional processing overhead at runtime.

For example, to temporarily transform the orientation of Element 1 in **14-7** on page 67 to match the Virtools Dev default coordinate system, perform the fol-

lowing steps.

1. Apply **Set Orientation** to Element1 and open the Edit Parameters dialog box.
2. Set **Referential** to --NULL-- to ensure that orientation calculations are performed relative to the orientation of the world.
3. Leave **Up**, **Dir**, and **Local Up** set to the default values.
4. Set Local Dir to (**X**=1, **Y**=0, **Z**=0) to inform Virtools Dev that the element's forward facing direction is the element's local positive X-axis.

Element1 now faces in the same direction as the world's positive Z-axis. Now, when you direct Virtools Dev to move the element forward in the default orientation (in the positive Z-axis direction relative to the world coordinate system), the element moves forward as expected.

32 EXAMPLE TRANSFORMATIONS

The essential data for each coordinate system is stored in a Local matrix and a World matrix respectively. The local matrix and the world matrix are both 4 * 4 matrices of the form show in **32-1**.

32-1 A coordinate system matrix

Xx	Xy	Xz	0	Right
Yx	Yy	Yz	0	Up
Zx	Zy	Zz	0	Dir
Xo	Yo	Zo	1	Position

where:

(X,Y,Z)x = The world space coordinates of the X axis of the local coordinate system expressed as a unit vector (the first three rows of the first column)

(X,Y,Z)y = The world space coordinates of the Y axis of the local coordinate system expressed as a unit vector (the first three rows of the second column)

(X,Y,Z)z = The world space coordinates of the Z axis of the local coordinate system expressed as a unit vector (the first three rows of the third column)

(X,Y,Z)o = The world space coordinates of the origin of the local coordinate system (the first three columns of the fourth row)

For each axis, the axis is defined as a unit vector (a vector of length 1) in the world coordinate system.

The values contained in the fourth column are constant.

In the **Parameter Debugger**, this matrix is displayed as:

[Xx, Xy, Xz, 0][Yx, Yy, Yz, 0][Zx, Zy, Zz, 0][Ox, Oy, Oz, 1]

[Right][Up][Dir][Position]

32.1 Translation

A translation with respect to the local coordinate system is expressed as:

32-2 Translation matrix

1	0	0	0
0	1	0	0
0	0	1	0
Xt	Yt	Zt	1

where:

(X,Y,Z)t = The translation vector that defines how far the target entity moves relative to the target entity's current position.

32.2 Rotation

32.2.1 About Local X Axis

A rotation about the local X axis is expressed as:

32-3 Matrix for rotation about X axis

1	0	0	0
0	Cx	Sx	0
0	-Sx	Cx	0
0	0	0	1

where:

Cx = the cosine of the angle of rotation about the X axis

Sx = the sine of the angle of rotation about the X axis

32.2.2 About Local Y Axis

A rotation about the local Y axis is expressed as:

32-4 Matrix for rotation about Y axis

Cy	0	-Sy	0
0	1	0	0
Sy	0	Cy	0
0	0	0	1

where:

Cy = the cosine of the angle of rotation about the Y axis

Sy = the sine of the angle of rotation about the Y axis

32.2.3 About Local Z Axis

A rotation about the local Z axis is expressed as:

32-5 Matrix for rotation about Z axis

Cz	Sz	0	0
-Sz	Cz	0	0
0	0	1	0
0	0	0	1

where:

Cz = the cosine of the angle of rotation about the Z axis

Sz = the sine of the angle of rotation about the Z axis

32.3 Scale

A scale factor is expressed as:

32-6 Scale factor

Sx	0	0	0
0	Sy	0	0
0	0	Sx	0
0	0	0	1

where:

Sx = the scale factor along the X axis

Sz = the scale factor along the Y axis

Numerics

2D Frames, creating 42
2D Point, see Point
2D Screen coordinates, see Coordinate syatem
2D Vector, see Vectors
3D Frames, creating 42
3D Layout 39
  camera navigation tools 43
  creation tools 42
  exploring 40
  selecting pivot point 41
  selection tools 40
  snapping elements 41
  top toolbar 39, 40
  transformation tools 40
3D Vector, see Vectors

A

Absolute coordinate system, see Coordinate system
Acronyms 28
Aggregation 86
Animations
  adding to a character 134
API 79
Arrays 95
  creating 48
Attributes 31, 127
  adding to an element 143
  Attributes Manager 127
  shortcuts 128
Axis, see Referential
Axis, selecting, see also Referential 41
Axis,constraining 41

B

Behavior 57
  adding outputs (bOuts) 146
  Building Block (BB) 107
    adding to a script 146
    adding to CMO 46
    attaching to element 46, 137
    processing 113
    symbols 107
  graph (BG) 114
  input (bIn) 108
  link (bLink) 108
  link delays 139
    editing 139
    showing/hiding 51
  loops 103
  output (bOut)
  processing 97, 99
  reusing 114
  scripts 107
Behavior Building Block (BB), see Behavior
Behavioral Engine 13, 101, 102
BG, see Behavior
bIn, see Behavior
bLink, see Behavior
bOut, see Behavior

C

C, Custom Dialog Box 111
Cameras
    activating at Scene start 137
    creating 42, 136
    default 136
    in rendering process 75
    navigating 43, 134
    selecting 40
    switching dynamically 146
    targeting 138
Characters
    adding to a Scene 134
    controlling 139
CKClass 84
Class hierarchy, class system 83
Compositions (CMO) 57, 91
    exporting to VMO 149
    saving 141
    sharing with others 150
Coordinate system 59
    2D 59
    3D 60
    local/relative 63
    world/absolute 64
Create Web Page 150
Creation tools
    3D Layout 42
    Level Manager 48
Curves, creating 42

D

Data organization 83
Depth of field 75
Depth sort 76
DirectX5 80
DirectX7 80
Document conventions 27
Documentation 23

E

Element 57
Event log, displaying 53

F

Frame, process loop 97
Frames per Second (FPS) 98
Further resources 23

G

Grids, creating 42
Groups 95
    creating 48, 145
    selection 40
Guides, reference and screen 42

H

Help, getting 23

I

IC, see Initial Conditions

Inheritance 85
Initial Conditions (IC)
    resetting 54
    setting 48
Interactivity
    creating 46
    managing 49

L

Left toolbar
    3D Layout 40
    Level Manager 48
Level 73, 92
    switching mode 48
Level Manager 47
    left toolbar 48
    top toolbar 47, 48
Lights, creating 42
Linking 50, 139
Local coordinate system, see Coordinate system
Local parameters, see Parameters

M

Materials 87
    creating 42
    self illuminating 133
Matrix 71
Measurement, units of 61
Media
    adding to CMO 46
    classifying 83
    importing 132
    sharing, see Sharing elements
Menus, description 37
Mesh sharing, see Sharing elements
Meshes 86
Messages 99, 113
    exploring 50
Metric system 61

O

OpenGL 80
Organizing data 83
Orientation 65, 66

P

Parameters 117
    changing type (pType) 147
    links (pLink) 117
    name 30
    name (pName)
    operations (paramOps) 121
    shortcuts 120
    showing local showing
    target 109
    This 119
    type (pType)
    value (pValue)
paramOps, see Parameters 121
Particles 129
Places 73, 94
    creating 48
Plane, see Referential

Plane, selecting, see also Referential 41
Play/Pause 54
    switching between modes 141
pLinks (parameter links), see parameters
pName (parameter name), see Parameters
Point
    2D 59
    3D 62
Portals 94
    creating 42
    management 94
Preferences, general 40
Priority 104
    showing/hiding 51
Process loop 97
Processing Building Blocks (BBs) 113
pType (parameter type), see Parameters 30
pValue (parameter value), see Parameters

Q

Quick start 129

R

Referential 64
    selecting axis or plane 41
Relative coordinate system, see Coordinate system
Render engine
    customizing 79
Rendering 79
    order 76
    process 75
Reset IC 54
Resources, BB and data 45
Rotate, see Transformations

S

S, Settings (BB) 111
Scale 61
    see also Transformations
Scenes 73, 93
    creating 48
    profiling 53
    rendering 98
    switching mode 48
Schematic 49
    exploring 50
    top toolbar 50
Screen coordinates, see Coordinate system
Scripts
    creating 48
    debugger 51
    showing/hiding header 50
    visualizing 49
Selecting, in 3D Layout 40
Selection 40
    locking 40
    mode 40
Sharing elements 88
Snapshot, taking 39
Space, segmenting in a CMO 94
Specialization 86

Status bar 53

T

Target parameter, see Parameters
Targetable 110
Textures 87
    creating 42
This, see Parameters
Time, segmenting in a CMO 93
Top toolbar
    3D Layout 39
    Level Manager 47
    Schematic 50
Trace mode, activating/deactivating 51
Transformations 69
    tools, see 3D Layout
Translate, see Transformations
Tutorials 129

U

Unit scale 61
User guide
    how to use 17
    organization 27

V

V, Variable BB 111
Vector2d, see Vectors
Vector3d, see Vectors
Vectors 62
Virtools Dev
    Authoring Application 13
    Behavioral Engine 13
    installing 17
    managers 13, 99
    MiniSite 129
    Render Engine 14
    screen at start-up 35
    Web Player 14, 150
VMO 149

W

Windows, moving/resizing 36
Workset, creating 48
World coordinate system, see Coordinate system
Worlds 73

Z

Z Buffering 75